Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado

Mestrado em Engenharia Informática

Pesquisa Segura de Dados em Redes de Sensores

Nuno Alexandre de Freitas Boavida (nº 27248)

Lisboa, Portugal
2010

Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado

Pesquisa Segura de Dados em Redes de Sensores

Nuno Alexandre de Freitas Boavida (nº 27248)

Orientador: Prof. Doutor Henrique João Domingos

Dissertação apresentada na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa para a obtenção do Grau de Mestre em Engenharia Informática.

Lisboa, Portugal
2010

Agradecimentos

Não obstante a individualidade da autoria da presente dissertação, dedico este espaço para referir todos os aqueles que contribuiram para o desenvolvimento da mesma, e os quais nunca é demais realçar a sua ajuda.

Ao Professor Doutor Henrique João Domingos, que, na minha visão de aluno e orientando, não poderia ter realizado um melhor trabalho de orientação durante todo o tempo de estudo teórico, aplicação prática e produção bibliográfica do tema da dissertação.

Ao Centro de Informática e Tecnologias da Informação (CITI), pertencente à Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, pela integração em actividades de investigação mediante a atribuição de uma bolsa científica e de acesso a um espaço de trabalho partilhado com outros vários alunos e investigadores.

A todos os meus colegas de curso, com os quais, ao longo dos passados anos, investi inúmeras horas dispendidas em estudos e projectos, com especial enfoque para o João Cartaxo, Pedro Bernardo, Nuno Luís, Gonçalo Martins, Pedro Sousa, Ricardo Martins, Amável Santo, entre outros que, apesar de não serem enunciados, não serão esquecidos.

Aos meus pais que, apesar das minhas persistentes ausências de casa, sempre senti de ambos um enorme apoio, presença e preocupação.

Aos meus amigos mais próximos, dos quais pude desfrutar da sua companhia e apoio durante dissertação, nomeadamente ao João Faria, Miguel Carvalho, Daniel Rolo, João Mendes, à Maria João Teixeira, e entre outros não menos importantes.

Aos vários utilizadores da *mailing list* da comunidade de desenvolvimento TinyOS pelas dúvidas debatidas e esclarecidas a pedido de outros, e pela ajuda a mim dirigida por parte dos Sr.s Bharat Naik, Tomislav Lipic, Mateus Santos Silva, Tomsy Paul, Vimal Kumar, que se disponibilizaram ao responderem ao meu pedido de ajuda.

A todos, os meus sinceros agradecimentos!

Sumário

O suporte de programação para Redes de Sensores sem Fios (RSSF) requer, cada vez mais, a consolidação de serviços intermédios (i.e. *middleware*) que forneçam abstracções interessantes para a produção de aplicações mais complexas do ponto de vista da heterogeneidade e da segurança. Estas abstracções devem incorporar garantias associadas a critérios de segurança das comunicações, tais como: autenticação, confidencialidade e integridade dos dados transmitidos e processados pela rede, bem como devem garantir intrinsecamente mecanismos de resistência face a ataques por intrusão.

A presente dissertação tem em vista propor, conceber, implementar e testar experimentalmente uma arquitectura de serviços *middleware* para organização de RSSF em torno de um conceito inovador: o de auto-organização baseada em vizinhanças lógicas seguras. Trata-se de disponibilizar uma abstracção de programação inspirada na noção de espaços de tuplos partilhados, aplicada ao desenvolvimento de aplicações para RSSF. A ideia tem em vista suportar aplicações orientadas à gestão e processamento interno seguro de dados (que se poderiam designar por *secure data-centric ad-hoc sensor networked applications*). Com base na pilha de serviços proposta, pretende-se elevar o conceito de proximidade física (mais associado aos critérios usuais de auto-organização das RSSF) para uma noção de proximidade lógica. Esta noção permite refinar ou optimizar processos de auto-organização orientada pelas aplicações, partindo de critérios que podem estar associados a propriedades de segurança e requisitos de heterogeneidade subjacentes ao nível dos serviços *middleware* a disponibilizar.

Este trabalho possui como motivação principal colmatar a lacuna de segurança existente em aproximações ao paradigma de vizinhanças lógicas, presentes na investigação recente das RSSF. Com base em propostas que introduziram e implementaram recentemente essa noção, estabelecida pela organização da rede com base em espaços de tuplos partilhados, pretende-se estender a ideia para a noção de "Vizinhanças Lógicas Seguras" (VLS). A principal contribuição do trabalho passa por permitir a simplificação da produção de aplicações críticas de segurança, possibilitando a definição e utilização das RSSF como repositórios confiáveis de sequências elementares de campos tipificados, vistos como conjuntos de espaços seguros de tuplos, definidos e utilizados de forma transparente pelas aplicações. Foi implementado e testado um protótipo expe-

rimental, concebido na linguagem NesC e em ambiente TinyOS, do qual se validaram as métricas de latência nas comunicações, coerência dos espaços de tuplos e consumo de energia, e que, adicionalmente, foi colocado em vários ambientes hostis de teste a ataques de retransmissão, omissão, integridade e comportamento arbitrário.

Palavras-chave: Redes de sensores sem fios (RSSF), Segurança em RSSF, Pilha de segurança para RSSF, Espaços de Tuplos Partilhados, Vizinhanças Lógicas, Vizinhanças Lógicas Seguras.

Abstract

The programming support for Wireless Sensor Networks (WSN) requires, even more, the consolidation of intermediate services (i.e. middleware) that provide interesting abstractions to produce more complex applications on the view of heterogeneity and security. These abstractions should incorporate guarantees related to communications' security criteria, such as: authentication, confidentiality and integrity of data transmitted and processed by the network, as well intrinsically ensure resistance mechanisms against attacks by intrusion.

The present dissertation aims to propose, develop, implement and experimentally test an architecture of middleware services for WSN organization around an innovative concept: of self-organization based on secure logical neighborhoods. This is to provide a programming abstraction inspired in the concept of shared tuple spaces, applied to the development of WSN applications. The idea is to support applications targeted to secure data management and internal processing (which could be designated as secure data-centric ad-hoc sensor networked applications). Based on the proposed stack of services, it is intended to raise the concept of physical proximity (more associated to the usual criteria of WSN self-organization) onto a notion of logical proximity. This concept allows to refine or optimize processes of self-organization driven by applications, based on criteria that may be associated with security properties and heterogeneity requirements subjacent to the level of the middleware services to provide.

This work's main motivation is to fill the security gap in existing approaches to the logical neighborhoods paradigm, present in the recent WSN research. Based on proposals that have recently presented and implemented this concept, established by the network's organization based on shared tuple spaces, it is wanted to extend the idea to the concept of "Secure Logical Neighborhoods" (SLN). The main contribution of the work is to allow the simplification of critical secure applications' production, enabling the definition and usage of WSN as trusted repositories of elementary sequences of typified fields, seen as sets of secure tuple spaces, defined and used transparently by applications. An experimental prototype was implemented and tested, developed in NesC language and on TinyOS environment, from which were validated the metrics of

communications' latency, tuple spaces' coherence and energy consumption, and was, additionally, placed in several hostile environments to test against attacks of retransmission, omission, integrity and arbitrary behavior.

Keywords: Wireless sensor networks (WSN), WSN security, Security stack for WSN, Shared Tuple Spaces, Logical Neighborhoods, Secure Logical Neighborhoods.

Conteúdo

Lista de Figuras

Lista de Tabelas

1

Introdução

A evolução das redes de sensores sem fios (RSSF), associada à (i) disponibilidade de alguns sistemas operativos de referência, (ii) normalização dos suportes de comunicação e (iii) evolução das capacidades dos próprios dispositivos de processamento, permite hoje a utilização combinada de dispositivos heterogéneos, dotados de capacidade de monitorização e actuação sobre o meio envolvente, além do desempenho de tarefas mais simples de captura, pré-processamento e transmissão de eventos associados a monitorização de fenómenos físicos particulares.

As arquitecturas descentralizadas de redes de sensores tornam-se cada vez mais interessantes e são cada vez mais investigadas para possibilidade de suporte de aplicações críticas de *sense-and-react*. Estas aplicações podem envolver processamento intermédio coordenado ao nível da própria rede, podendo executar em ambientes eventualmente hostis. Nestes ambientes, onde a rede tem condições mais vulneráveis do ponto de vista de vigilância ou gestão permanente, os nós de processamento (sensores e actuadores) podem ser objecto de maior probabilidade de intrusões físicas. Deste modo, a rede poderá passar a funcionar de forma incorrecta por acção de adversários que desencadeiem ataques externos (ao nível do sub-sistema de comunicação) ou que realizem, com sucesso, ataques por intrusão em um ou mais dispositivos.

Ao nível do sistema de comunicações, os atacantes podem desencadear ataques externos à rede, violando propriedades básicas de segurança, usualmente estudadas ao nível da seguirança das comunicações em redes [53], nomeadamente: (i) autenticação de nós e de dados transmitidos e processados ao longo da rede, (ii) integridade de dados, (iii) preservação de garantias de confidencialidade dos dados ou mensagens transmitidas, bem como (iv) detecção de retransmissão ilícita de mensagens.

Ao nível do processamento interno dos nós da rede, os atacantes podem desencadear ataques a diferentes níveis: (i) ao nível do sub-sistema de processamento do

encaminhamento ou de difusão de dados, (ii) ao nível de agregação ou processamento intermédios de dados transmitidos pela rede, ou (iii) ao nível de eventual adulteração de processamento local, por intrusões físicas e alteração da lógica de processamento num ou mais nós, o que pode levar o adversário a controlar a rede, em maior ou menor grau. Este último tipo de ataques é particularmente crítico, por poder permitir ao adversário ter acesso a segredos criptográficos de maior ou menor duração, o que, por sua vez, anula as eventuais garantias de segurança das comunicações estabelecidas por protocolos e métodos criptográficos.

Não obstante as dificuldades de suporte de aplicações críticas de segurança em que as dimensões anteriores de tipologias de ataque estão em jogo, as redes de sensores têm conhecido um interesse sempre crescente, quer ao nível da investigação, quer ao nível do seu potencial de aplicação. O potencial das RSSF torna-as particularmente interessantes na criação de ambientes de forte descentralização de processos de monitorização ou de actuação, com baixos custo de investimento e de gestão operacional.

A descentralização com base em redes de sensores poderá permitir a adopção de técnicas de pré-processamento ou agregação localizada e classificada de dados e eventos, melhorando o consumo dos recursos e reduzindo a latência da comunicação desses dados, o que é vantajoso face às aproximações centralizadas que estejam permanentemente a receber uma grande quantidade de eventos originados em todos os nós de monitorização. Este aspecto é particularmente relevante para minimizar o consumo energético da rede e dos seus dispositivos, uma vez que os custos energéticos da comunicação são determinantes face aos custos de processamento local.

Igualmente, a descentralização do processamento com base em RSSF permite a utilização de uma tecnologia em que o custo por nó de processamento é baixíssimo, o que leva à oportunidade de construir redes de maior ou menor densidade e de maior ou menor distribuição, em função de necessidades específicas das aplicações.

No entanto, o desenvolvimento de aplicações fortemente descentralizadas com base em RSSF torna-se naturalmente mais complexo, requerendo a colocação da lógica de controlo e das assumpções de segurança no interior da rede, e exigindo uma coordenação adequada das actividades de processamento e agregação de dados, internas aos dispositivos. Estes requisitos relevam a importância de garantir comunicação fiável e segura na rede, bem como requerem a necessidade de introdução de mecanismos de resiliência face a ataques por intrusão, que devem ser suportados de forma intrínseca aos serviços de middleware que suportam as aplicações.

1.1 Motivação da dissertação

A presente dissertação insere-se na problemática da concepção e disponibilização de serviços de *middleware*, tendo como motivação a criação de ambientes de programação que facilitem a produção de aplicações seguras para RSSF eventualmente heterogéneas. A orientação da elaboração da dissertação tem em vista conceber, implementar e testar experimentalmente uma pilha de serviços *middleware* para organização de RSSF em torno de um conceito inovador: o conceito de auto-organização baseado em vizinhanças lógicas seguras (VLS). Através desta noção, disponibiliza-se um ambiente de programação inspirado no modelo de espaços de tuplos aplicado ao domínio das redes de sensores e à programação de aplicações orientadas ao processamento interno e seguro de dados (ou *secure data-centric ad-hoc networked applications*).

Com base na pilha de serviços a propor para o anterior suporte de *middleware*, pretende-se elevar o conceito de proximidade física (implicitamente associado aos critérios usuais de auto-organização das RSSF) para uma noção de proximidade ou vizinhança lógica. O critério de vizinhança lógica visa refinar ou optimizar o processo de auto-organização com base em critérios definidos pela semântica das aplicações e que pode incluir a definição das próprias propriedades e condições de segurança requeridas. O trabalho possui como motivação principal colmatar a lacuna de segurança existente em aproximações ao paradigma de "vizinhança lógica", na investigação recente das RSSF [18, 42, 56, 57]. Com base nas anteriores propostas que introduziram e implementaram a noção de vizinhança lógica a partir da organização funcional ou semântica da rede, pretende-se estabelecer a noção de "vizinhança lógica segura", que se estabelece complementarmente com base em critérios e propriedades de segurança, incluindo adicionalmente funcionalidades de reorganização face ao dinamismo da rede.

Como contribuição principal do trabalho, pretende-se assim simplificar a produção de aplicações com requisitos de segurança, possibilitando a definição e utilização das RSSF como repositórios confiáveis de sequências elementares de campos tipificados. Estes, são vistos como conjuntos de espaços seguros de tuplos, definidos e utilizados de forma transparente pelas aplicações. Nas restantes secções desta introdução, apresenta-se uma visão introdutória sobre as principais características das redes de sensores sem fios, a implicação dessas propriedades na implementação dos critérios de segurança, a forma como se estruturam os serviços numa pilha de arquitectura, e, finalmente, abordam-se os objectivos e contribuições desta dissertação.

1.2 Redes de Sensores sem Fios

Os mais recentes avanços tecnológicos dos sistemas *micro-electrical-mechanical* (MEMS) permitiram o desenvolvimento de nós sensores de baixo custo e energia, constituintes das RSSF [11]. Estas redes são compostas por um elevado número de nós sensores, densamente implantadas no cerne do fenómeno a capturar, ou muito próximo do mesmo) [11]. Um aspecto particular é o seu modo de execução num esforço cooperativo entre os elementos da rede, objectivando maior fiabilidade de dados [14], melhor consumo energético e mais rápida obtenção de eventos. O estabelecimento das RSSF, geralmente, não obriga a um posicionamento determinado, permitindo a sua colocação de forma livre e aleatória, aspecto que torna essencial que os protocolos e técnicas nelas aplicadas não careçam de capacidades próprias de auto-organização [12].

O conjunto de características principais que diferenciam as RSSF das redes *ad-hoc* englobam: um número de nós sensores superior em várias ordens de magnitude, são mais propensas a falhas, a topologia da rede é frequentemente alterada, baseiam-se em paradigmas de comunicação *broadcast*, e apresentam maiores limitações nas capacidades dos dispositivos que as formam) [11]. As características desses dispositivos possuem aspectos determinantes, associados à sua reduzida dimensão: a sua capacidade de processamento e memória são bastante limitados, e o seu poder energético reduzido. Este último, por sua vez, é uma das mais importantes restrições nos dispositivos constituintes das RSSF, visto os nós sensores possuírem fontes de energia bastante limitadas, de baixo poder e quase sempre insubstituíveis) [11]. Para que alcance um elevado nível de qualidade de serviço, os protocolos e técnicas empregues devem ter em conta, primeiramente, o consumo energético. Esse recurso é mais penalizado pelas operações de comunicação, só depois pelas operações de processamento, e finalmente, pelas operações de amostragem de eventos físicos.

A topologia das RSSF pode ser caracterizada por ser *flat*, onde todos os nós sensores possuem o mesmo papel, ou *clustered*, nas quais existem nós principais de *clusters* que controlam a comunicação [33]. Em níveis um pouco mais ascendentes, são empregues três tipos de paradigmas arquitecturais em RSSF: cliente-servidor, código móvel e espaços de tuplos [33]. As arquitecturas de *software* são suportadas pela plataforma base disponibilizada pelo sistema operativo incluído no dispositivo, onde se preza a óptima compatibilização das primitivas de comunicação com as capacidades hardware [50]. Como exemplos de sistemas operativos para RSSF encontra-se o TinyOS, o Contiki [1], o Mantis [2], entre outros. Devido às suas características apresentadas, surge um grande número diverso de possíveis utilizações das RSSF, incluindo aplicações militares, ambientais, de saúde, residenciais, educacionais entre outras [11, 33, 50].

1.3 Estruturação de Serviços em Redes de Sensores sem Fios

Com a existente diversidade de mecanismos, protocolos e serviços que têm sido propostos para redes de sensores, é necessário que os mesmos sejam organizados de forma lógica, tendo em vista a consolidação de arquitecturas de *middleware*, de base reutilizável por parte de aplicações. Estas arquitecturas podem permitir desde formas de auto-organização baseadas em descoberta de nós (tendo por base critérios de proximidade ao nível físico no que diz respeito aos requisitos de cobertura da rede e do suporte de comunicação rádio), até critérios de refinamento de organização da rede dirigida por requisitos do nível aplicacional.

No estado da arte das redes de sensores, não existe porém uma estruturação de serviços unificada ou normalizada [33], como acontece com as arquitecturas de redes de computação convencionais. Nas arquitecturas que têm sido propostas na investigação das RSSF, a estruturação de serviços é feita em maior ou menor especificidade com base em cada aplicação particular. Deste modo, definem-se tipicamente conjuntos de serviços (como módulos de suporte de aplicações específicas) que utilizam como denominador comum as especificações de *standards* para nível MAC ou *Data-Link* (ou ligação de dados) habitualmente associados às RSSF (e.g., 802.15.4 [10], Zigbee [13]).

Esta estruturação difere da visão clássica do modelo de referência do tipo OSI [59] visto os requisitos das aplicações de RSSF serem também muito diversos em relação aos requisitos de implementação e adaptação aos ambientes em que executam [33]. Na sua generalidade, é visível, na figura 1.1, que o sistema operativo apenas engloba os serviços do nível MAC, e que os suportes *middleware* a desenvolver devem proporcionar uma adaptação adequada das aplicações ao suporte básico de comunicação.

Na estruturação de serviços em RSSF descrita na figura 1.1, a camada física suporta as primitivas de comunicação baixo-nível que permitem o envio e recepção de dados através do meio de transmissão rádio. Imediatamente acima, a camada *Data-Link* é apenas composta pela porção MAC, não existindo concretamente uma noção alargada de funcionalidade associada habitualmente ao nível *Logical Link Control* (controlo da ligação lógica de dados), típica nas implementações da pilha OSI ou de alguns protocolos de pilhas convencionais (e.g. TCP/IP), nomeadamente para suporte de operações de multiplexagem/desmultiplexagem de tramas, ou de controlo de integridade e de fluxo de dados. Por outro lado, a camada MAC numa arquitectura de RSSF actua de acordo com topologias definidas da rede, definindo um protocolo de acesso e partilha do meio físico e de comunicação rádio entre os participantes [33].

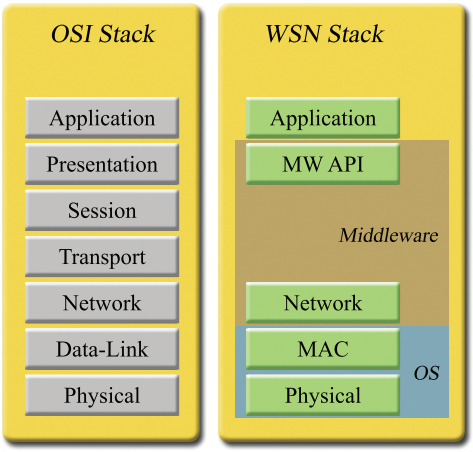


Figura 1.1: Estruturação de referência do modelo OSI (*esquerda*) em comparação com a estruturação geral de serviços em RSSF (*direita*).

1.4 Segurança em Redes de Sensores sem Fios

As características inerentes às redes de sensores sem fios proporcionam vários desafios ao nível da implementação de serviços e mecanismos de segurança, visto estes últimos introduzirem custos pesados na implantação das propriedades desejadas. Muitas das técnicas de segurança estabelecidas nas redes tradicionais são inaplicáveis nas redes de sensores estudadas neste trabalho. Os aspectos essenciais de segurança em redes de sensores englobam o estabelecimento de chaves criptográficas, confidencialidade e autenticação, privacidade, comunicação e encaminhamento de dados robustos e, finalmente, resiliência a intrusões [48].

Os dispositivos usados nas redes de sensores possuem limitações importantes em relação à capacidade de processamento, memória e energia disponível. Estas limitações inviabilizam a possível utilização de abordagens de protocolos de segurança baseados em processos criptográficos que são habitualmente exigentes de recursos mínimos de computação e de memória (primária ou secundária). Tal é o caso de protocolos criptográficos baseados em criptografia de chave pública ou em assinaturas digitais com algoritmos de chaves públicas usando métodos assimétricos convencionais, ou serviços de autenticação e confidencialidade de dados que exijam a geração, gestão ou manutenção de pares de chaves e cadeias de certificados de chave pública [49].

A utilização de processos criptográficos muito exigentes do ponto de vista de complexidade do processamento é igualmente uma questão decisiva em relação à necessidade de minimizar o consumo de energia. Este factor é essencial pelo facto de ditar o tempo de vida de cada dispositivo e da rede em geral, pelo que se devem utilizar preferencialmente métodos criptográficos com características de processamento leve

e estruturas ou mensagens de dados de dimensões reduzidas, uma vez que a comunicação é responsável pela maior parte do consumo de energia. A implementação de mecanismos de segurança como módulos distintos dos restantes componentes é, geralmente, uma abordagem incorrecta que proporciona pontos de ataque específicos [48], devendo abarcar todos os níveis arquitecturais do sistema, numa perspectiva orientada à concepção integral dos serviços da pilha.

Com efeito, as ameaças de segurança em redes de sensores podem surgir em diferentes níveis da pilha, tal como apresentada na anterior figura 1.1. No capítulo 2, dedicado a uma visão sobre trabalho relacionado, apresentar-se-á uma visão estruturada de tipologias de ataques aos diversos níveis da pilha de estruturação de serviços de *middleware* para RSSF. A caracterização destes ataques, numa visão integrada, é um aspecto essencial para o estabelecimento de modelos de adversário que devem formar a base das contra-medidas associadas aos serviços de segurança subjacentes aos vários níveis da pilha.

1.5 Objectivos e Contribuições da Dissertação

As preocupações e requisitos anteriormente introduzidos estão associados aos objectivos e contribuições da presente dissertação. O objectivo principal tem em vista conceber, implementar e testar experimentalmente uma pilha de serviços *middleware* para organização de RSSF em torno do conceito de auto-organização baseado em vizinhanças lógicas seguras. Como contribuição relevante, é proposto um ambiente de programação inspirado no paradigma de espaços de tuplos partilhados, aplicado ao domínio das redes de sensores e à programação de aplicações orientadas ao processamento interno e seguro de dados (ou secure data-centric ad-hoc networked applications). Esta noção concretiza-se como um ambiente de programação implementado sobre uma pilha de serviços, estruturada da seguinte forma:

- Foi integrado um serviço de comunicação segura sobre o suporte de comunicação básico rádio, que disponibiliza mecanismos de autenticação, confidencialidade, integridade de dados. A camada de comunicação básica incorpora um protocolo seguro de comunicação de dados capaz de resistir a ataques externos ao meio de comunicação. Esta camada possui características adequadas de baixo consumo de energia, utilizando criptografia simétrica leve, adequada para implementação em redes de sensores e assinaturas baseadas em códigos de integridade (MAC-Signatures ou Message-Authentication-Code Signatures), utilizando funções de síntese de baixo custo computacional. O funcionamento desta camada, que foi

concretizada por recurso a uma implementação inicialmente existente [32], requer ainda a integração de um mecanismo eficiente de estabelecimento de chaves criptográficas (inexistente na referida implementação), capaz de implementar esquemas de refrescamento seguro de chaves com base em bons indicadores de eficiência e baixo consumo energético.

- Sobre a anterior camada são suportados, ao nível rede, os seguintes serviços:
  - Um serviço de auto-organização da rede que assegura condições optimizadas de cobertura, do ponto de vista de uma estrutura de comunicação segura, par-a-par.
  - Um serviço de encaminhamento com suporte para difusão segura de dados, com características de resiliência capazes de resistir a ataques por intrusão em nós sujeitos a ataques bizantinos, por omissão ou alteração das mensagens difundidas. Este serviço cria e adopta rotas de encaminhamento alternativas, escalonadas por processos aleatórios, como resistência à exploração de ataques em nós específicos da rede, que são descartados por detecção de rotas que possam estar a ser sujeitas a atacantes intermédios.
  - Um serviço e uma API de pesquisa de dados. Este serviço é suportado pelo anterior protocolo de difusão de dados, permitindo disponibilizar um mecanismo de interrogações sobre dados, a partir de qualquer nó e sobre qualquer nó da rede.
- Sobre o anterior suporte de interrogação e difusão de dados foi implementada uma camada que cria a abstracção de vizinhança lógica segura, como expressão da organização da rede com base em espaços seguros de tuplos partilhados, definidos por critérios estabelecidos a partir de uma API com a devida expressividade.

A anterior pilha de serviços, como suporte integrado de *middleware* disponibilizado por um ambiente de programação para desenvolvimento de aplicações, eleva o conceito de proximidade física (que está na base dos critérios usuais de auto-organização das RSSF) para uma noção de proximidade ou vizinhança lógica segura. O critério de vizinhança lógica, que pode ser estabelecido a partir da API disponibilizada aos programadores de aplicações, permite refinar ou optimizar o processo de auto-organização. Tal é possível baseando-se em critérios definidos pela semântica das aplicações e pela definição de requisitos e propriedades a estabelecer a partir das mesmas, como propriedades associadas à noção de espaços seguros de tuplos usados como como repositórios confiáveis de sequências elementares de campos tipificados, definidos e utilizados de forma transparente pelas aplicações.

1.6 Organização do Documento

Após o presente capítulo introdutório, onde foi exposta uma visão inicial da solução concretizada, face aos desafios e requisitos impostos pelas redes de sensores sem fios e suas características próprias, o restante documento está organizado da seguinte forma: O Capítulo 2 apresenta uma pilha de referência da estruturação de serviços de segurança da solução, e realiza uma análise *bottom-up* à mesma, abordando as mais determinantes soluções nos serviços de comunicação segura, de gestão e estabelecimento de chaves, de encaminhamento e de difusão segura de dados, e de suporte à programação orientada a espaços de tuplos, relacionadas com o âmbito deste trabalho.

No Capítulo 3, é descrito o modelo e arquitectura da pilha de serviços das vizinhanças lógicas seguras, sendo que para cada um dos níveis de serviços enunciados na anterior secção 1.3, foi escolhida uma aproximação relacionada para inclusão no sistema final implementado. São também apresentados os aspectos essenciais da forma como cada componente principal da pilha de serviços foi concebido, e ainda também como cada objectivo que o sistema concretiza foi alcançado.

O Capítulo 4 apresenta o cerne da implementação do sistema de vizinhanças lógicas seguras, descrevendo, não só o ambiente de desenvolvimento e simulação/emulação empregue no trabalho desta dissertação, bem como os componentes principais da pilha edificada que foram formalmente implementados. Adicionalmente, este capítulo reserva atenção ao módulo de segurança das comunicações e ao cenário de validação experimental da solução final.

O Capítulo 5 abarca a validação experimental da pilha implementada, descrevendo quais os testes realizados, bem como os objectivos, interpretação e análise dos resultados obtidos em cada um deles, terminando com uma síntese final de todas as simulações executadas.

Finalmente, é efectuada no capítulo 6 uma análise conclusiva do trabalho realizado, face às contribuições e objectivos introduzidos no primeiro capítulo, discutindo adicionalmente como a arquitectura proposta os alcança, e apresentando-se, por fim, algumas hipóteses em aberto que são consideradas como interessantes para trabalho futuro.

2

Trabalho Relacionado

Neste capítulo, é realizada uma análise a abordagens de referência na investigação das RSSF, discutindo-se os seus aspectos mais relevantes para a concretização da pilha de VLS implementada nesta dissertação. Este capítulo divide-se assim em secções correspondentes a cada nível da pilha a apresentar na secção 2.1: nível de comunicação segura 2.2, encaminhamento de dados 2.3, estabelecimento seguro de chaves 2.4, pesquisa e difusão segura de dados 2.5 e gestão de espaços de tuplos partilhados 2.6.

2.1 Pilha para Análise de Trabalho Relacionado

Na figura 2.1 apresenta-se uma pilha estruturada em múltiplas camadas, representativa de uma arquitectura de variados serviços (entre os quais de segurança, difusão de dados e espaços de tuplos) relacionada com o sistema desenvolvido nesta dissertação. A análise de trabalho relacionado centra-se entre os serviços base de comunicação (SO e MAC) e as aplicações, efectuando-se no presente capítulo uma análise *bottom-up* dessa mesma pilha. Tal estruturação por níveis é uma visão geral dos serviços e sua organização na arquitectura que concretizou na implementação da presente dissertação.

Inicialmente, são descritas as responsabilidades impostas aos serviços de comunicação segura ao nível da utilização dos serviços de controlo de acesso ao meio, visto assentarem directamente sobre estes últimos para os tornar seguros face a ataques de *replaying*. Para tal, necessitam de um esquema de gestão e estabelecimento de chaves criptográficas, formando assim as contra-medidas básicas de ataques às comunicações. Os serviços de encaminhamento são dos processos mais prejudicados pelos atacantes, pelo que a sua segurança recorre a mecanismos de resiliência para resistir a ataques de intrusão. Os serviços de difusão segura permitem realizar *broadcasts* seguros, modelo de comunicação bastante interessante para o *middleware* das VLS. Finalmente, o

Figura 2.1: Estruturação de Serviços de Segurança em RSSF e foco da análise (a vermelho).

middleware fornece a abstracção apresentada nesta dissertação por forma de um sistema de gestão de espaços de tuplos partilhados, disponibilizando às aplicações um conjunto de primitivas de programação de utilização dos mesmos.

2.2 Comunicação Segura em Redes de Sensores

Os ataques direccionados às RSSF podem ser efectuados com vista a prejudicarem determinados mecanismos de um certo nível da pilha de serviços. Torna-se então fulcral que os critérios de segurança e que a dependência com as camadas inferiores sejam tidos em conta em todos os níveis, de forma a evitarem-se ataques a uma dada camada por indução de ataques às inferiores. Assim, o ponto de partida para a implementação da segurança situa-se na camada adjacente ao suporte básico de comunicação fornecido pelo sistema de operação. Aborda-se então, nesta secção, a problemática da obtenção de comunicação segura em RSSF, apresentando-se variadas arquitecturas que assentam sobre plataformas base de comunicação.

2.2.1 Plataforma Base

Com a falta de maturidade das RSSF, ainda não se encontra estabelecida uma agregação de serviços e protocolos suficientemente generalizada para a maioria das aplicações. Tal obriga a que o desenvolvimento dessas aplicações se inicie num nível muito

próximo do nível das primitivas básicas de comunicação fornecidas pela plataforma base, intrínseca ao dispositivo de RSSF, e compreende o nível físico e o nível MAC.

A principal implementação de plataforma base de comunicação para RSSF é especificada pelo standard IEEE 802.15.4 [10]. O standard emprega como protocolo de nível MAC um modelo CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance), slotted ou unslotted, conforme se pretenda sincronizar as comunicações, ou não, pelo envio de *beacons* por parte de um coordenador, que divide o tempo de acesso em slots [41]. Tal parametrização permite adaptar a camada a aplicações que exijam maior fiabilidade ou tolerem falhas de comunicação. O acesso é controlado com base em pressupostos de justiça e eficiência no acesso ao meio de comunicação, tendo por base o procedimento CCA (Clear Channel Assessment, existente no nível físico), para determinar se o canal pode ser utilizado ou se se deve esperar um intervalo de tempo (backoff time), aleatoriamente calculado pelo emissor. Essa duração depende de um expoente de backoff , derivado do poder energético do dispositivo, favorecendo as condições mais precárias. É também mantida uma variável de tentativas realizadas. Ainda assim, é da responsabilidade das camadas superiores a alteração desses parâmetros, pelo que podem ser alvo de exploração por parte de atacantes.

Ao longo do tempo foram produzidas outras especificações normalizadas para níveis MAC de RSSF. No âmbito do Grupo 4 - IEEE 802.15 [54] e ZigBee Alliance [3], foi normalizada uma especificação que engloba requisitos de comunicação de redes baseadas em pequenos dispositivos rádio para baixo consumo energético. Na verdade, a especificação ZigBee [13] abrange outras camadas superiores de rede, empregando o standard 802.15.4 para especificar os níveis Físico e Data-Link. Não sendo uma implementação pura de nível MAC, o ZigBee define protocolos de comunicação que abarcam funcionalidades de inicialização, afiliação ou abandono de uma rede, sincronização com outros dispositivos, encaminhamento e segurança de frames [13].

2.2.2 Arquitecturas e Serviços de Camada Segura ao Nível MAC

As abordagens de comunicação segura podem vir incluídas numa implementação/especificação de vários níveis da pilha, como o ZigBee ou a norma 802.15.4, ou apenas relativas a esse nível cimeiro, tais como as que serão apresentadas nesta secção, assentando sobre implementações MAC do 802.15.4, ou do BMAC/ZMAC.

Para a realização do estudo dessas soluções, é particularmente interessante a análise de um conjunto das suas características, tais como o nível de segurança providenciado, o consumo de energia requirido e recursos exigidos por cada uma das aproximações. Nas seguintes subsecções, são apresentadas as características de cada uma dessas aproximações de camada segura de implementações do nível MAC.

2.2.2.1 Norma 802.15.4

A especificação 802.15.4 [10] permite a aplicação de mecanismos de segurança sobre as frames MAC, conforme as propriedades de segurança da suite escolhida pela aplicação, como controlo de acesso, confidencialidade, integridade e frescura. É deixada às camadas superiores a responsabilidade de produção e gestão de todo o material criptográfico, a autenticação de participantes nas comunicações e a protecção da frescura de pacotes. O nível MAC da norma 802.15.4 fornece os seguinte serviços:

- **Listas de Controlo de Acesso (ACL)**: para designar as origens confiáveis das *frames* recebidas, permitindo a filtragem de dados não autorizados;
- **Mecanismos de Cifra de Dados**: para confidencialidade das *frames*, mediante chaves simétricas partilhadas entre um par de nós, ou entre um grupo, armazenadas em cada entrada respectiva da ACL. No entanto, os *addresses* existentes em cada frame são mantidos em claro;
- **Serviço de Integridade de *frames***: para prevenir alterações às *frames*, pelo uso de MICs (*Message Integrity Code*) produzidos de blocos de *ciphertext* com a chave partilhada;
- **contadores de *frames* e contadores de sequência de chaves**: por forma a garantir a frescura de *frames* e evitar *replays* das mesmas.

Os mecanismos acima mencionados são utilizados conforme a escolha de um dos três modos: ***unsecured***, que não faz uso de nenhum dos mecanismos, dirigido para aplicações de baixo custo [41] e que não requerem segurança; ***secured***, onde são usados os mecanismos conforme a suite de segurança escolhida; e modo **ACL**, que emprega apenas o serviço de controlo de acesso possibilitado pelo uso das listas ACL.

O nome de uma *suite* de segurança, escolhida quando em modo *secured*, especifica o algoritmo criptográfico simétrico, o modo e o tamanho em bits do código de integridade [1]. O algoritmo de cifra empregue em qualquer *suite* é o AES (*Advanced Encryption Standard*), podendo ser usado com modo CTR, CCM ou CBC-MAC, e com estes dois últimos usando códigos de integridade com comprimento de 32, 64 ou 128 bits.

2.2.2.2 ZigBee

A camada segura da arquitectura ZigBee [13] assenta no *framework* básico de segurança da norma 802.15.4 para fornecer as propriedades de confidencialidade, controlo de acesso, integridade e frescura. A sua gestão encontra-se ao nível rede, controlada

[1] Note-se que o comprimento do código de integridade é menor ou igual ao tamanho do bloco do algoritmo de cifra e que não influencia o poder do algoritmo subjacente.

por um serviço fornecedor de segurança. A solução alcança um elevado nível de segurança, em detrimento do consumo de energia, permitindo o uso de diferentes chaves alteráveis e protegendo contra ataques de *replay*.

No ZigBee, o mecanismo de protecção contra ataques de *replaying* é composto pela manutenção dos contadores em ambos os extremos da comunicação, que são incrementados por cada mensagem recebida ou enviada. Esses contadores formam os *Initialization Vectors* (IV) usados nos métodos criptográficos. Para a suportar os refrescamentos criptográficos, os contadores são reiniciados em cada alteração da chave usada no mecanismo de cifra e na produção dos MACs. A sincronização é conseguida pelo envio do valor do contador com a mensagem, resultando em *overheads* de 8 bytes.

2.2.2.3 TinySec

O TinySec [32] é um módulo de segurança leve e genérico sobre a pilha TinyOS, para uma fácil integração em aplicações de RSSF. Esta solução fornece as propriedades de autenticidade, integridade e confidencialidade de dados. No entanto, não protege de ataques de *replaying*, justificando que tal deve ser feito em níveis superiores [32].

São disponibilizados dois modos de segurança, o TinySec-AE, que fornece as propriedades de autenticação e confidencialidade dos dados, e TinySec-Auth, garante apenas a autenticidade. O acréscimo na dimensão das mensagens do modo TinySec-AE é de 5 bytes, enquanto o TinySec-Auth acrescenta apenas 1 byte. A confidencialidade de dados é obtida através do método Skipjack com modo CBC, cujo IV de 8 bytes é transmitido em cada mensagem, para, adicionalmente, realizar a sincronização entre emissor e receptor. Este sistema de cifra de camada link é nativo do TinyOS, que, por sua vez, produz a chave secreta por defeito. A integridade dos dados advém do uso do mecanismo de cifra de blocos CBC-MAC, que produz códigos com 4 bytes de comprimento.

2.2.2.4 SPINS

O SPINS [49] é um conjunto de componentes orientado para ambientes com restrição de recursos e comunicações sem fios, constituído pelo SNEP, que fornece serviços base de segurança, e pelo μTESLA que permite efectuar *broadcasts* autenticados [2]. A confidencialidade é obtida através da cifra de blocos em modo CTR, baseada em contadores mantidos em cada *end-point*, modo que fornece também frescura fraca das mensagens. Para autenticação de dados, é usada a técnica CBC-MAC, reutilizando a cifra de confidencialidade dos dados. Adicionalmente, os códigos MAC permitem autenticar os participantes da comunicação.

[2] O protocolo μTESLA irá ser abordado neste documento na secção 2.5.

Relativamente à sincronização dos contadores entre emissor e receptor, estes são mantidos de uma forma independente, sendo incrementados em cada mensagem enviada ou recebida. De tal maneira, no SPINS a dimensão das mensagens é reduzida visto a sincronização não ser implícita, isto é, por não enviar o contador em cada envio de mensagem.

2.2.2.5 MiniSec

A solução MiniSec [38] associa propriedades básicas de segurança (confidencialidade, integridade, autenticação e frescura dos dados) com um consumo de energia reduzido. Esta arquitectura de comunicação emprega um mecanismo de cifra autenticada, através do método Skipjack com modo OCB (*Offset CodeBook*), que fornece confidencialidade e autenticidade numa só passagem, e usa como vector de inicialização IV um contador incremental. A sincronização desse contador é mantida entre emissor e receptor pelo envio de apenas alguns bits do mesmo, para que o requisito de frescura seja ultimado. O número de bits é parametrizável, permitindo equilibrar o consumo energético face ao nível de segurança.

O MiniSec permite elevada flexibilidade visto possuir dois modos de comunicação: *Unicast* e *Broadcast*, orientados, respectivamente, para transmissão ponto-a-ponto ou multi-ponto. No MiniSec-Unicast, a sincronização dos contadores é feita pelo envio de x bits do IV, procedendo-se a computações extra nas situações de incoerência. No modo MiniSec-Broadcast, são empregues épocas com sincronização fraca para detectar *replays* fora da respectiva janela de tempo, e filtros de Bloom para detectar retransmissões dentro da janela de tempo.

2.2.3 Discussão Sumária

A tabela 2.1 sintetiza uma comparação entre as arquitecturas de segurança deste capítulo tendo em vista os seus mecanismos e dimensões de mensagens, com respectivos incrementos face ao modelo base do TinyOS.

O nível de segurança do TinySec é reduzido, factor relacionado com o uso de uma chave única na rede, que facilita ataques de *masquerading*, e com a falta de protecção face a ataques de *replay* de mensagens. No entanto, o seu consumo de energia e requisitos de memória são reduzidos, não mantendo estado de emissor ou receptor e empregando MACs de dimensão reduzida. A sincronização entre *end-points* através do envio do IV nos pacotes TinySec-AE gera algum *overhead* no tamanho das mensagens, mas evita eventuais re-sincronizações. Realça-se a flexibilidade fornecida por esta arquitectura ao fornecer dois modos de segurança: autenticação e confidencialidade, ou

Protocolo	Mecanismos			Mensagem	Overhead		Total	Aumento face TinyOS
	Cifra	Autenticacao	Frescura	Payload	Pacote	Segurança		
TinyOS	-	-	-	24	12	-	36	-
ZigBee	S	S	Fraca	24	20, 24, 32	4, 8, 16	44, 48, 56	22.2%, 33.3%, 55.6%
TinySec	S/N	Nenhuma	N	24	17	5	41	13.9%
SNEP	S	S	Fraca	24	20	8	44	22.2%
MiniSec	S	Forte ou Fraca	S	24	15	3	39	8.3%

Tabela 2.1: Tabela sumária de comparação entre arquitecturas de comunicação segura ao nível dos seus mecanismos de segurança e *overheads* na dimensão de mensagens. Baseada em [15, 38]

apenas autenticação.

No ZigBee é obtido um maior nível de segurança do que no TinySec, visto não se limitar ao uso de uma chave única e fornecer protecção contra ataques de *replay*. No entanto, o consumo de energia e a latência são penalizados, incorrendo em *overheads* na comunicação, devido ao facto do IV ser acrescentado às mensagens enviadas na sincronização dos contadores. Outro ponto negativo do ZigBee é a manutenção do estado por emissor, exigindo maior capacidade de memória, principalmente em casos de grande escala.

O SNEP segue uma abordagem inversa das restantes arquitecturas, visto efectuar uma sincronização implícita. Tal processo resulta num consumo energético mediano e a um reduzido *overhead* na comunicação. No entanto, esse tipo de sincronização torna o SNEP pouco recomendado para situações de elevada perda de pacotes, obrigando a frequentes re-sincronizações. O seu nível de segurança é elevado, visto fornecer segurança semântica derivada do uso dos contadores, protegendo dos ataques de *eavesdropping*.

Das soluções de comunicação segura nesta secção apresentadas, o MiniSec é a que, de forma mais equilibrada, concretiza as propriedades de segurança (confidencialidade, autenticidade, frescura) face ao consumo energético e às exigências de recursos. Tal é conseguido devido ao reduzido *overhead* introduzido nas mensagens pela técnica implementada de sincronização fraca, obrigando à manutenção do estado nos *endpoints* com complexidade espacial constante. Fornece flexibilidade ao incluir protecção a ataques de *replay* em dois padrões de comunicação, ponto-a-ponto e multi-ponto, através da aplicação de, respectivamente, sincronização forte e fraca. Finalmente, outro aspecto muito positivo da solução MiniSec é que existe uma implementação e o seu código fonte está publicamente disponível para a plataforma TelosB [4].

2.3 Encaminhamento Seguro de Dados

A importância do encaminhamento seguro surge com a necessidade da comunicação *out-of-range* ser baseada no *forwarding* efectuado por um ou mais nós intermediários, onde cada elemento da rede actua como um *router*. Adicionalmente, muitas soluções possuem a tendência de incluir todos os nós disponíveis como participantes no encaminhamento, tornando-os intrínsecos nesse processo [30, 55]. Esses factores, conjuntamente com as características do meio de comunicação e dos dispositivos, originam várias oportunidades de ataque. Nesta secção, avalia-se o conjunto de ameaças ao nível do encaminhamento de dados, e correspondentes serviços e técnicas de segurança que objectivam a sua supressão.

2.3.1 Ataques ao encaminhamento

Os ataques à segurança são classificados em passivos ou activos. Um ataque passivo consiste no acesso à informação capturada de um fluxo de dados. Aqui encontramos os ataques de *Eavesdropping*, com posterior *Release of Contents* e Análise de Tráfego (Padrões). Visto não destabilizarem o processo de encaminhamento, a detecção destes ataques é difícil. Não obstante, são facilmente mitigados com o uso de mecanismos de cifra ou partição dos dados por múltiplos caminhos e posterior junção no destino [55].

Os ataques activos objectivam a quebra do mecanismo de encaminhamento, a obtenção de autorização ou de autenticação, ou até o controlo da rede através do fabrico de pacotes falsos ou da alteração ou supressão de pacotes legítimos a circular na rede. Um ataque activo ao encaminhamento pode ser classificado como externo (o atacante não possui acesso a informação sensível) ou interno (atacante possui acesso a informação sensível devido a uma intrusão a um ou vários nós) [55]. Um ataque externo é efectuado por nós maliciosos que não pertencem à rede.

Numa visão mais aprofundada, a sua tipologia especifica-se em três classes, conforme a fase de execução do protocolo de encaminhamento na qual o ataque é realizado: ataques ao processo de descoberta de rotas, ataques ao processo de selecção de rotas, e ataques ao próprio processo de encaminhamento após o estabelecimento de rotas. A descrição desses ataques é apresentada de seguida.

2.3.1.1 Ataques ao Processo de Descoberta de Rotas

Falsa Informação de Rotas: através de anúncios de informação falsa ou supressão de pacotes de descoberta, a construção das tabelas de encaminhamento pode tornar-se incorrecta.

Ataques de *Rushing*: antecipando a recepção dos pacotes de descoberta legítimos, um atacante pode injectar pacotes adulterados ou fabricados, para introduzir rotas incorrectas.

2.3.1.2 Ataques ao Processo de Selecção de Rotas

Ataques de Difusão de HELLO: um atacante com elevado alcance de transmissão, do tipo *laptop-class*, pode dar-se a conhecer a um grande número de nós, iludindo-os de que o emissor se encontra na sua vizinhança directa (*one-hop*), interferindo com protocolos de localização;

Ataques de *Sinkhole*: um nó malicioso tenta suscitar a preferência de outros nós legítimos por ele, com o objectivo de atrair na sua direcção as rotas dos seus vizinhos, convergindo o tráfego para ele. Apesar de afectarem só uma parte da rede, podem originar ataques de encaminhamento selectivo, ou auxiliar ataques de *Wormhole*.

Ataques de *Wormhole* [17]: através do uso de um canal *out-of-bound* de baixa latência, origina-se a falsa aparência de que a rota entre dois nós maliciosos distantes é uma melhor escolha, para que sejam escolhidos como intermediários de nós legítimos. Com a colocação de um deles próximo a uma estação, é possível controlar parte significativa do encaminhamento.

Ataques de *Sybil* [44]: com o anúncio de várias identidades distintas, um nó malicioso pode personificar múltiplos nós, aumentando a possibilidade de ser escolhido pelos vizinhos como link de encaminhamento. É um ataque direccionado a protocolos com mecanismos multi-rota.

2.3.1.3 Ataques ao Processo de Encaminhamento

Ataques de *Blackhole*: são acções de não reencaminhamento das mensagens recebidas por um atacante, evitando que continuem a ser propagadas pela rede. Existe a variante selectiva em que o atacante escolhe descartar apenas algumas mensagens;

Ataques de *Spam*: geração de um número elevado de mensagens não solicitadas e inúteis com o objectivo de desgastar a largura de banda e a de energia de nós legítimos. O atacante pode conseguir particionar a rede de modo crasso se atacar nós próximos do *sink*.

2.3.2 Abordagens de Encaminhamento Seguro

Nesta subsecção, apresenta-se um leque de protocolos fundamentais no âmbito do encaminhamento seguro em RSSF. Essencialmente, estas aproximações fazem uso de

criptografia leve para mitigar ataques externos e possuem mecanismos base de resiliência activa, como encaminhamento multi-rota, a reestruturação topológica, ou ambos.

2.3.2.1 INSENS

O INSENS [19] é um protocolo de encaminhamento tolerante a intrusões, que ultima as propriedades de confidencialidade, integridade, autenticação e frescura, no caso de ocorrerem poucas capturas. As tabelas de *routing* indicam múltiplos caminhos, preferencialmente independentes e disjuntos, entre um nó e uma *base station*. A construção das tabelas é realizada numa acção conjunta entre todos os elementos da rede e as estações, considerando a minimização do peso computacional sobre os nós.

O protocolo efectua um *flooding* de pedidos autenticados por uma cadeia sequencial numérica ao estilo do μTESLA [49] para determinar a topologia da rede. Os *feedbacks* (respostas) são autenticados por MACs baseados numa chave partilhada entre cada nó e estação, remetendo ao método SNEP [49], e seguem o caminho inverso ao percorrido pelos pedidos, mantido em cadeias criadas ao longo da disseminação. Com a recepção dos *feedbacks*, numa acção consciente da topologia da rede, a estação constrói as tabelas e verifica inconsistências introduzidas por atacantes. A resiliência multi-rota é obtida pela criação de dois caminhos distintos entre cada nó e uma estação.

No processo de encaminhamento, os nós trocam dados com a estação num modo *Unicast*, cujas mensagens tendencialmente percorrem os caminhos estabelecidos. Apesar de ser uma técnica simples, cuja segurança se baseia na veracidade das tabelas, com as quais se confrontam as identidades dos nós emissores, a sua eficácia depende do número de nós comprometidos nos caminhos e da porção de nós hierarquicamente abaixo dos comprometidos na fase inicial de construção das tabelas.

2.3.2.2 Secure Leach

Para redes hierárquicas (*cluster-based*), o LEACH [25] é um protocolo com objectivos de distribuição equilibrada do consumo energético, que emprega uma rotação aleatória dos elementos principais dos grupos. Esses elementos, denominados *cluster-heads*, são o ponto de comunicação dos elementos do grupo com o exterior, pelo que podem realizar agregação de dados. O SecLEACH [45] surge como uma extensão geral de segurança do LEACH, introduzindo segurança nas suas fases de execução.

A sua contribuição passa por tornar seguras as comunicações e interacções de suporte ao dinamismo hierárquico, através de uma solução pré-distribuição aleatória de chaves proposta. O protocolo fornece confidencialidade, autenticidade, integridade e frescura de dados. O seu modo de execução por rondas *setup* (descoberta das partilhas de chaves e seu estabelecimento) e *steady-state* (coordenação de comunicação por *time*

schedules, agregação realizada pelo *head* e envio para uma estação) é adequado ao dinamismo das RSSF *clustered*. No entanto, o protocolo apenas é dotado de um mecanismo de resiliência, para efectuar reorganizações de topologia [45].

A comunicação ocorre apenas entre estações e *heads* ou entre *heads* e *nodes*, pelo que os *heads* são pontos de ligação da rede, tornando-se atraentes a ataques. Para cada ronda, é realizada a fase de *setup* e múltiplas fases de *steady-state*. São empregues mecanismos simétricos de cifra e autenticação nos *links* entre nós de um *cluster*, e *broadcasts* autenticados suportados por cadeias de chaves *one-way* ao estilo do μTESLA para tornar segura a comunicação entre *heads* e estações.

2.3.2.3 Clean Slate

A solução Clean Slate [46] compreende um protocolo hierárquico para encaminhamento de dados em RSSF, tendo em conta a segurança e a eficiência energética, com os objectivos de prevenir, detectar, recuperar e resistir a ataques a este nível. As tabelas de *routing* são resultantes de uma fase preliminar de agrupamento recursivo hierárquico de nós. O determinismo do algoritmo previne alterações ao encaminhamento, caracterizado como lógico, concretizado por saltos entre grupos. Esse método é eficiente ao nível funcional, mas despreza a eficiência do encaminhamento físico, onde o envio de um pacote para um grupo pode envolver múltiplos *hops* na rede.

Na fase inicial, os nós são agrupados desde um estado singular até se formar um grupo que contenha todos. Tendencialmente, grupos de tamanho similar irão fundir-se, fazendo melhor uso do espaço de endereços. A descoberta é concretizada pelo envio de pedidos autenticados pelo certificado do nó emissor, produzido antes da implantação da rede por uma *Network Authority*. Em cada iteração, é actualizada uma *Group Verification Tree* (GVT) de cada nó. Uma GVT é uma *Hash Tree* contendo os IDs dos grupos, e nas folhas os IDs dos nós, e permite a um grupo autenticar o seu ID e o seu tamanho no agrupamento com outros grupos. Este método de agrupamento determinista, conhecido por todos, mantém a unicidade dos IDs dos nós e permite futuras autenticações, possibilitando a detecção de manipulações ao processo.

O seu método "Honeybee"de remoção de participantes maliciosos é peculiar, pois um nó que anuncie uma detecção de falha ou captura de outro nó, ambos são afastados da rede. Contrariam-se assim os ataques de *Slandering*, pois um nó subvertido não consegue afastar da rede um nó legítimo (por anúncios de reputação negativa) sem que seja também removido. As entradas dos nós removidos nas tabelas podem ser usadas para outros nós válidos, pelo que a remoção de nós subvertidos não prejudica significativamente os futuros encaminhamentos. A reaplicação da fase inicial de Setup permite a adição de novos nós e a adaptação a variações das condições da rede.

2.3.3 Discussão Sumária

A seguinte tabela 2.2 sintetiza as propriedades dos protocolos estudados em relação ao suporte de contra-medidas face à tipologia de ataques considerados e que estão na génese da modelação das hipóteses de adversário.

	Ataques ao Encaminhamento							
Protocolo Estudado	Ataques Externos	Informação Falsa	Ataques Rushing	HELLO Flood	Ataques Sinkhole	Ataques Wormhole	Ataques Sybil	Ataques Blackhole
INSENS	Total	Total	Parcial	Total	Total	Parcial	Parcial	Parcial
SecLEACH	Total	Total	-	-	-	-	-	-
Clean Slate	Total	Total	Parcial	Total	Total	Parcial	Total	Parcial

Tabela 2.2: Nível de protecção fornecido pelas soluções estudadas face a ataques ao encaminhamento em redes de sensores.

O INSENS tolera intrusões e consequentes ataques de supressão de pacotes, desde que um dos múltiplos caminhos criados por um nó não sofra um ataque. A difusão dos pedidos de material topológico depende da segurança do μTESLA e permite, com os "keyed MACs", mitigar ataques de *Denial-of-Service* por pedidos repetidos. No caso de captura de um nó, o atacante tem acesso apenas a uma chave secreta, eliminando a possibilidade de obter chaves dos vizinhos ou de outros nós. As cadeias parentais conferem resiliência no processo de descoberta, permitindo resistir a *diverts* de *feedbacks*, que acabam por tomar outro caminho em direcção à estação. No entanto, incorre em *overheads* significativos no tamanho das mensagens, e o seu modo de contenção de tráfego pode não ser apropriado, pois leva à contaminação dos vizinhos abaixo na cadeia dos nós subvertidos, podendo alcançar uma grande porção da rede.

Relativamente ao SecLEACH, é obtida comunicação par-a-par segura, atingindo as propriedades de confidencialidade, autenticação, integridade e frescura de dados. É empregue um esquema de pré-distribuição aleatória de chaves suportando a comunicação e interacções entre nós seguras, novidade orientada para redes hierárquicas. No entanto, um ponto negativo revela-se ao nível da protecção face a intrusões, não contemplando medidas contra ataques activos internos. Destaca-se o mecanismo de resiliência activa de reinício das rondas e reestruturação da topologia da rede.

O Clean Slate é uma aproximação segura de elevado nível, cuja natureza determinista, que apesar de dar a conhecer a qualquer tipo de participante o método de operação do protocolo, previne comportamentos incorrectos. A redundância da comunicação e a aplicação de métodos criptográficos, em conjunto com a detecção de nós subvertidos pelo uso das árvores de fusão e de GVT, tornam esta aproximação bastante completa. A técnica "HoneyBee"de repulsa de intrusos permite também contrariar ataques de *slandering* a nós legítimos, a não ser que o atacante esteja disposto a

ser sacrificado. O Clean Slate é o mais completo dos apresentados nesta secção, sendo que engloba mecanismos de resiliência de encaminhamento por múltiplas rotas e de reestruturação da topologia da rede.

2.4 Estabelecimento Seguro de Chaves Criptográficas

A gestão de chaves criptográficas é um aspecto essencial na segurança das comunicações, e no caso das RSSF, a segurança é ainda mais crucial, visto serem colocadas em locais hostis, pouco ou nada supervisionados, estando mais susceptíveis a riscos.

Existem três tipos de esquemas de estabelecimentos de chaves criptográficas: distribuição, acordo e pré-distribuição de chaves [29]. O primeiro é pouco viável para RSSF, pois exige a presença e conectividade a um servidor confiável, que se torna um ponto central de ataque. O segundo esquema depende de métodos criptográficos assimétricos, que se são pouco praticáveis em RSSF devido às suas exigências de recursos. O terceiro tipo de estabelecimento de chaves é o único considerado praticável nestas redes, numa escala significativa, e sem ponto único de falha [29]. Nesta secção do documento, serão estudados modelos interessantes de esquemas de pré-distribuição aleatória de chaves.

2.4.1 Esquema aleatório *Pairwise*

O esquema de chaves *pairwise* representa um modelo em que cada nó possui uma chave estabelecida distinta com todos os restantes nós [16]. Este esquema alcança o nível máximo de segurança face a capturas de chave, sendo que numa ocorrência de tal, apenas um *link* seguro é comprometido. É visível que é um modelo bastante exigente em termos de memória, não permitindo escalar a redes de dimensão elevada.

Uma modificação a esse modelo extremo é o esquema aleatório *pairwise*, proposto por [16]. Neste modelo há um relaxamento do número de chaves partilhadas, reduzindo-se esse número m para apenas um valor necessário a obter-se uma probabilidade mínima de partilha de chaves, dado o número de nós na rede. O seu modo de execução passa por atribuir, de forma determinista e anterior ao *deployment* da rede, a correspondência de cada nó a um grupo de outros nós, para que quando se efectuem os *handshakes* criptográficos se descubram as partilhas de forma não-reveladora. Este método determinista permite obter propriedades interessantes de segurança, tais como: autenticação node-to-node [3], resiliência perante intrusões, replicação ou introdução de nós, e revogação descentralizada.

[3]Um protocolo possui a propriedade de autenticação *node-to-node* se qualquer nó conseguir diferenciar a identidade dos nós com quem comunica.

Adicionalmente, a revogação de chaves é concretizada por um algoritmo distribuído de votos públicos, que não exige uma autoridade central confiável. Um nó, ao detectar a presença de um nó comprometido, divulga um voto público a anunciar essa detecção. Ao se atingir um determinado *threshold* de votos contra um dado nó, as ligações com o mesmo são quebradas. Apesar de fornecer um maior nível de segurança e cobertura que os esquemas de pré-distribuição aleatórios, não goza da propriedade de escalabilidade.

2.4.2 Esquema de estabelecimento com distribuição aleatória

O esquema de distribuição aleatória de chaves proposto por Eschenauer e Gligor [20], denominado esquema básico, baseia-se na probabilidade da partilha de chaves entre os nós de um grafo aleatório, e emprega um simples protocolo de descoberta de chaves partilhadas para suportar a sua distribuição, revogação e restabelecimento. O protocolo divide-se em três fases: pré-distribuição de chaves, descoberta de chaves partilhadas e descoberta de chaves *path-key*.

A fase de pré-distribuição de chaves ocorre antes da implantação da rede no ambiente, e nela é atribuído, a cada nó, um subconjunto aleatório oriundo de uma ampla *pool* de chaves. Garante-se que, com um tamanho reduzido do chaveiro, a probabilidade de dois nós partilharem, pelo menos, uma chave é elevada. Na inicialização da rede, cada nó determina quais os seus vizinhos directos com quem partilha, pelo menos, uma chave, por um processo que não revela as chaves. Tal é feito pelo simples envio e comparação dos identificadores das chaves, ou, numa via mais segura, pelo envio e resposta correcta de desafios cifrados pelas chaves que o nó emissor possui. Após se descobrirem os vizinhos com chaves em comum, é possível estabelecer ligações de comunicação seguras entre cada par deles. A terceira fase de descoberta de chaves *path-key*, permite criar as ligações seguras entre vizinhos directos que não partilham chaves, com o auxilio de intermediários (outros vizinhos directos comuns).

O modelo suporta um processo de revogação de chaves relativas a um nó, realizada através do anúncio cifrado de uma lista composta por essas mesmas chaves, recorrendo a um controlador confiável. Pode incorrer nalguma latência visto alguns *links* directos poderem desaparecer, obrigando à repetição da segunda fase, ou até da terceira, inclusive. Finalmente, este esquema tem em consideração o restabelecimento de chaves, para os casos de expiração de validade associados a essas mesmas. O processo é local a cada nó e não requer a presença de um controlador confiável, removendo a chave e restabelecendo os *links* seguros dependentes da mesma através dos processos de descoberta de chaves partilhadas ou também de descoberta de chaves *path-key*.

2.4.3 Esquema q-Composite

O esquema q-Composite [16] é uma modificação do método de pré-distribuição aleatória apresentado acima em 2.4.2, com diferenças no tamanho da *pool* de chaves e no número de chaves partilhadas entre dois nós, denominado q, necessário para formar um *link* seguro entre eles, ao invés de uma única chave como na solução base.

O método de operação deste esquema é bastante similar ao seu original, sendo que a diferença situa-se no número verificado de chaves partilhadas entre nós. A motivação do esquema passa por simplesmente aumentar o *overlap* das chaves partilhadas entre cada dois nós, com o objectivo de uma superior resiliência à captura de dispositivos. A chave de um *link* é a combinação das múltiplas chaves partilhadas (K) pelo par de nós. A sua resiliência é bastante elevada, face a ataques de pequena escala, obrigando o atacante a efectuar ataques de larga escala (mais caros e mais detectáveis) [16].

Devido à multiplicidade das chaves partilhadas necessárias para se criar uma ligação segura entre vizinhos, o valor do parâmetro S (tamanho da *pool* de chaves) torna-se crucial para a qualidade de segurança do método. Se S for demasiado elevado, a probabilidade de partilha das q chaves seria insuficiente; ou se S for demasiado reduzido, as chaves K de cada ligação irão convergir para a mesma.

No entanto, sendo possível verificar a legitimidade de outros nós nestes esquemas de distribuição aleatória, não é possível autenticar-se a identidade de um nó com quem se comunica [16]: um nó, ao partilhar um grupo de chaves com múltiplos nós, não consegue diferenciar a comunicação com eles, pois todos têm acesso válido ao *link* seguro.

2.4.4 Discussão Sumária

A tabela 2.3 sintetiza as principais características dos esquemas de estabelecimento de chaves desta secção, bem como alguns dos mecanismos centrais que incorporam.

Esquema Estudado	Nível de Escala	Nível de Cobertura	Nível de Resiliência	Revogação por NA	Restabelecimento por NA	Autenticação *node-to-node*
PW Básico	Reduzido	Elevado	Elevado	-	-	Sim
PW Aleatório	Baixo	Alto	Alto	Não	-	Sim
Distr. Aleatória	Elevado	Baixo	Baixo	Sim	Não	Não
q-Composite	Elevado	Reduzido	Elevado*	Sim	Não	Não

Tabela 2.3: Tabela sumária de comparação entre modelos de estabelecimento de chaves e suas características principais. (* - para ataques de pequena escala)

Como se pode verificar na tabela apresentada, os esquemas *pairwise* obtêm uma elevada resiliência, derivada de uma elevada cobertura na partilha de chaves entre

pares de nós. Tal resulta numa menor eficácia de ruptura de segurança, pois ao se quebrar um *link* seguro, nenhum ou poucos serão também comprometidos. Ainda assim, desfrutam de um processo de revogação de nós sem a dependência de uma *Network Authority*, tornando o processo autónomo e diminuindo os pontos de ataque. No entanto, a sua capacidade de escalabilidade é muito reduzida, exigindo para casos de redes de grandes dimensões, elevados recursos de armazenamento, pelo que estas abordagens são direccionadas para redes de reduzida dimensão.

Por outro lado, os esquemas de pré-distribuição aleatória não proporcionam uma autenticação *node-to-node*, reduzindo o seu nível de resiliência, afectado também pelo facto de não impedir replicação ou introdução de nós. No entanto, permitem uma maior capacidade de escala face aos esquemas *pairwise*. Desta forma, o esquema aleatório q-Composite destaca-se dos restantes por ser um modelo mais equilibrado e adequado por adicionar às características do esquema básico aleatório um elevado nível de resiliência face a ataques de pequena escala, mais comuns em RSSF.

2.5 Pesquisa e Difusão Segura de Dados

Nesta secção, apresenta-se um conjunto de sistemas de pesquisa e difusão de dados, cuja importância para as RSSF é materializada com a sua robustez, escalabilidade e consumo energético. Apesar de alguns não contemplarem segurança ao nível das comunicações, incluem mecanismos de resiliência implícita, contornando assim alguns ataques externos.

2.5.1 TinyDB

O TinyDB [39] é um sistema de obtenção de dados por interrogações (*queries*) que abstrai do programador muita da complexidade da implementação e gestão de pesquisas. É composto por: um processador de *queries* que optimiza a eficiência energética; mecanismos de introdução automática de redundância e evitação de áreas problemáticas da rede; e difusão eficiente de *queries* e dados de retorno.

É fornecida uma interface de *querying SQL-based*, permitindo concorrência de *queries*, e monitorização da rede através das mesmas. A sua linguagem de *query*, o TinySQL, engloba algumas das cláusulas principais do SQL, o SELECT, FROM, WHERE e GROUP BY, suportando as operações relacionais de selecção, junção, projecção e agregação. No seu retorno, os tuplos são produzidos em concordância com uma computação estruturada em épocas de amostragem, definidas pelas cláusulas SAMPLE PERIOD ou ONCE, para minimizar o consumo energético. Disponibiliza também outros tipos de

queries, adequadas para RSSF, como as baseadas em eventos, as de agregação e as baseadas em tempo de vida.

A transmissão de *queries*, submetidas pelas *base stations*, baseia-se numa difusão ao longo de uma árvore de encaminhamento derivada de um processo de "*beaconing*". Nesse processo, um nó executa a *query* localmente (se conseguir produzir resultados da mesma) e/ou posteriormente transmite-a aos seus filhos se estes puderem executá-la. Tal verificação, baseada numa estrutura proposta pelos autores, *Semantic Routing Tree* (SRT) [4], torna o encaminhamento eficiente, evitando a difusão desnecessária de queries que não se apliquem numa sub-árvore de nós.

2.5.2 Directed Diffusion

O Directed Diffusion [27] é um protocolo de difusão *hop-by-hop* para RSSF, que se adapta ao ambiente envolvente e a falhas na rede. A sua execução baseia-se em interacções localizadas, sendo a propagação e agregação de dados feita por cada vizinhança. A difusão é centrada nos dados, onde a informação possui o formato de pares atributo-valor, dirigida a uma metodologia "*query based*", e pode ser *cached*, transformada ou agregada por cada nó. Esta falta de noção global *end-to-end* contribui para a robustez, escala e eficiência energética deste protocolo.

A rede é acedida pela disseminação de interesses (pedidos de captura de eventos) nos nós *sink*, e os nós *source* respondem com dados dos eventos capturados do meio físico. A difusão inicial de interesses em baixa taxa é exploratória, e tenta alcançar todos os nós que capturarão os eventos requisitados. Nessa disseminação, repetida enquanto a tarefa se mantiver activa no *sink*, são formados gradientes, figurativos da ligação entre cada nó e indicam o sentido e taxa de transferência do *link*. A transmissão de dados baseia-se numa distribuição por múltiplos caminhos com qualidade selectiva, criados pela difusão de interesses. Os dados são capturados do exterior em concordância com a taxa definida pelos gradientes por eles estabelecidos. No processo, é mantida uma *cache* de dados em cada nó, evitando ciclos e gastos desnecessários de recursos.

Eventualmente, a qualidade do caminho preferencial que os dados percorrem será "estimulada" pelo *sink*, aumentando a taxa de captura e transferência dos eventos. A escolha do caminho estimulado baseia-se em regras locais a cada nó (e.g. menor delay, perdas) e é concretizada pelo envio de reforços positivos (interesses com maior taxa). Desta forma, o protocolo é bastante sensível às alterações de qualidade nos links. É também possível reduzir-se a qualidade de caminhos, com o envio de interesses de baixa taxa, para que seja possível repararem-se caminhos degradados ou desconecta-

[4]: Uma Semantic Routing Tree (SRT) permite a cada nó detectar se os seus filhos produzirão resultados para uma dada *query*

dos, resultantes de falhas, detectados e realizados de forma autónoma por cada nó.

2.5.3 μTESLA

O μTESLA [49] é um protocolo de *broadcasts* autenticados para RSSF, e é uma adaptação desse método existente para redes tradicionais, o TESLA [47]. Para tal, emprega somente mecanismos criptográficos simétricos, uma revelação de chaves menos frequente e dividida em épocas, e uma gestão de cadeias de chaves efectuada apenas pelos que iniciam a difusão de *queries* (as *base stations*). Fornece as propriedades de integridade, autenticidade, e frescura de dados.

O seu modo de operação divide-se numa fase inicial de *bootstrap* de uma cadeia de chaves, e num ciclo de *broadcasts* autenticados por cada chave ao longo dessa cadeia, e posteriores revelações de cada uma. Todo este método cíclico pode ser repetido até a cadeia ser totalmente revelada, e assenta no pressuposto de que uma chave apenas será revelada quando todos os receptores já possuirem o pacote por autenticar. O tempo de emissão é dividido em épocas, cada uma correspondendo ordenadamente (de 1 até n) a uma chave na cadeia, usada para produzir um MAC que acompanha os pacotes emitidos. Tendo conhecimento das épocas através de uma sincronização fraca entre emissor e receptores, é possível descartarem-se pacotes que não verifiquem o pressuposto de segurança assumido.

Após o intervalo de tempo desde o ínicio de uma época (d), parametrizável, a sua chave correspondente é difundida pelo nó emissor, por forma a que os receptores possam obter para verificar a autenticação de mensagens dessa época que possuam em *buffer*. Tal veracidade é resultado da aplicação da função global sobre a chave recebida, que resulta na chave da época anterior, até alcançar a chave difundida na fase de *bootstrap*, K_0, autenticando-se o pacote com sucesso. O protocolo é flexível na medida em que permite autenticar pacotes anteriores à epoca de uma chave recebida, mesmo sem se possuirem as chaves anteriores.

2.5.4 Secure Directed Diffusion

Para mitigar as falhas de segurança do Directed Diffusion [27], o Secure Diffusion [58] possui como objectivos manter elevada a qualidade da difusão de informação legítima, na presença de nós comprometidos e numa rede não particionada, e conter nas vizinhanças de nós maliciosos o tráfego falso por eles injectado.

São acrescentados três tipos de chaves para concretizar a autenticação de nós: cadeias de chaves *one-way* para *broadcast* seguro de interesses autenticados pelo *sink*, ao

estilo do μTESLA [49]; chaves de célula *location-binding* [5] (LBK) para autenticar a origem dos dados capturados pelos *sources*; e chaves de vizinhança estabelecidas por um esquema *pairwise* para autenticar gradientes e criar *links* seguros entre vizinhos. Estes mecanismos actuam em conjunção com os originais, sem lhes subtrair a robustez, escala e eficiência energética [6], e a sua fase de *bootstrap* assume que esses métodos são eficientes, executando mais rapidamente que uma intrusão a cada aparelho.

O mecanismo de reforços original é atraente para os atacantes, possibilitando alterações nos caminhos preferenciais para *path influence*, e consequentes ataques. Uma alteração para o tornar seguro passa por realizar reforços não só com base na qualidade de recolha dos dados, mas também na autenticidade dos mesmos [7]. O papel do *sink* é incrementado para assistir qualquer nó da rede no seu reforço local, através da técnica de reforço selectivo, que classifica os nós conforme a autenticidade dos dados por eles capturados. Os nós gerem a sua lista de vizinhos numa estratégia rotacional de *round-robin*, convergindo o caminho preferencial reforçado para um livre de intrusos.

2.5.5 Discussão Sumária

O μTESLA é adequado para *broadcasts* autenticados em RSSF, empregando mecanismos de segurança simétricos, mas a sua escala é limitada pela necessidade de se manter em *buffer* os pacotes por autenticar, e requer sincronização fraca entre emissores e receptores [15]. As cadeias de chaves conferem flexibilidade, pois com uma chave é possível autenticar pacotes de épocas anteriores das quais não se possuam as correspondentes chaves.

O TinyDB é um sistema eficiente na utilização dos recursos de comunicação e energia, suportado por um processador de queries e um protocolo eficiente de interrogações. Permite ao programador abstrair da complexidade da pesquisa e obtenção de informação em RSSF, para se focar no desenvolvimento da aplicação. O seu ponto negativo é a falta de segurança, alcançável com a inclusão de componentes adicionais do TinyOS (p.ex. TinySec).

O Directed Diffusion é um protocolo de pesquisa e encaminhamento que desfruta das propriedades de robustez, escalabilidade e eficiência energética, devido ao facto dos seus métodos serem aplicados localmente. É sensível a alterações de qualidade na rede, devido à natureza reactiva do seu mecanismo de reforços de caminhos, conferindo-lhe uma elevada resiliência face a acções de *Denial-of-Service* [30].

[5] O tipo de chaves *Location-Binding Keys* foi uma novidade apresentada pelos autores do Secure Diffusion.

[6] Exceptuando os overheads impostos pela adição de segurança.

[7] O facto de um nó anunciar que se encontra num caminho de reduzido *delay* não é uma boa base de segurança, pois essa promoção pode ser falsa

O Secure Diffusion objectiva a manutenção da qualidade de difusão na presença de nós maliciosos, não sendo prejudicado pelo tráfego adulterado por eles injectado. Permite também a repulsa de atacantes dos caminhos preferenciais, adaptando o mecanismo de reforços. A segurança é adicionada sem denegrir as características do original, mantendo os métodos localizados, robustos e escaláveis. No entanto, o mecanismo de localização das chaves geográficas é pouco plausível, pois exige recursos significativos de processamento e energia.

2.6 Suporte de Programação Orientado a Espaços de Tuplos Partilhados

Nesta secção, apresentam-se plataformas *middleware* baseadas em espaços de tuplos partilhados, que generalizam e tornam transparentes à aplicação a gestão das operações de baixo nível. Os espaços de tuplos permitem um desacoplamento emissor-receptor favorável à concorrência e assincronismo dos métodos executados orientados à mobilidade. A maioria do trabalho desta secção tem como alvo as RSSF, não obstante as restantes soluções para outras redes serem interessantes.

2.6.1 Lime

O LIME [43] é um *middleware* para redes *ad-hoc* de suporte ao desenvolvimento de aplicações que exibam mobilidade física, lógica ou ambas, sendo uma extensão ao modelo Linda [23] para ambientes móveis. A grande diferença é o espaço de tuplos global ser fisicamente distribuído pelos nós e logicamente particionado e partilhado conforme a conectividade entre eles.

Outros factores do LIME de suporte à mobilidade englobam: manutenção transparente do contexto global da rede, conseguida pelo modo de distribuição e partição do espaço de tuplos e pela definição das partilhas transitórias de espaços conforme a conectividade; submissão de reacções sobre espaços de tuplos, para que um conjunto de acções execute em *background* à rede, aquando de uma correspondência de um tuplo a um template; noção de espaço federado (junção dos espaços de tuplos de vários nós) e disponibilização de operações para adição e remoção de nós aos mesmos.

O acesso e manipulação dos espaços de tuplos são feitos mediante as principais operações do Linda: **out**, **rd** e o **in**, bem como as não bloqueantes **inp** e **rdp**, cuja semântica original é preservada. A selecção de tuplos é realizada pela correspondência do seu conteúdo com um *template* fornecido na pesquisa. Os espaços de tuplos no LIME são complementados com uma noção de localização para permitir implementar

as suas migrações e capacidade de reacção a um dado estado.

Sendo a mobilidade um factor principal nesta plataforma, o uso dos espaços de tuplos permite o desacoplamento espacial e temporal da comunicação entre processos, não sendo mandatório a presença do emissor e receptor no mesmo instante de tempo da operação. Outro factor que contribui para a mobilidade é a manutenção transparente do contexto global da rede, conseguida através do particionamento e distribuição do espaço de tuplos do Linda em vários espaços, cada associado a um nó, e da definição das formas de partilha transitória desses espaços conforme a conectividade.

O acesso ao contexto global é fornecido por uma interface de espaço de tuplos existente em cada nó, representando os tuplos que esse *host* pretende tornar públicos aos restantes nós de uma comunidade. As primitivas empregues no acesso ao espaço de tuplos global são baseadas nos métodos do Linda, e a sua semântica original é preservada. Desta forma, é transmitida ao nó a ilusão de se ter um espaço de tuplos local que contém todos os tuplos pertencentes à comunidade, sem ser necessário o conhecimento explícito da mesma.

Adicionalmente ao mecanismo de partilha transitória, para suportar adição e remoção de nós aos espaços federados, surgem duas sequências de operação: *engagement*, para receber novos nós móveis que se juntam ao espaço de tuplos federado, sendo uma operação atómica; e *disengagement*, para a saída de um nó móvel, resultando na remoção dos seus tuplos partilhados existentes noutros nós.

O modelo Linda também é extendido tendo em vista a elevada propensiação ao dinamismo do ambiente derivado da mobilidade dos hosts. Uma reacção é definida por um fragmento de código que especifica as acções a realizar aquando de uma correspondência do conteúdo de um tuplo com um *template/pattern* ocorre no espaço de tuplos. A execução das reacções é concorrente com o restante processamento do nó em questão, e podem ser de dois tipos: reacções fortes, restritas à mesma localização (mesmo *host* ou agente); ou reacções fracas, com escopo sobre um espaço de tuplos federado, isto é, vários espaços de tuplos locais, com o objectivo de detectar alterações no mesmo.

2.6.2 Hood

O Hood [57] é uma abstracção de programação para RSSF onde um nó identifica subconjuntos de vizinhos, mediante certos critérios, e com eles partilha dados. Cada nó possui acesso a uma estrutura de dados local, recipiente do *caching* de atributos que lhe interessam, fornecidos por nós vizinhos. É um paradigma pró-activo, onde os dados fluem num padrão de comunicação *many-to-one* e os receptores recolhem os atributos em que estão interessados. São considerados vizinhos directos (*one-hop*) e o

seu método envolve o *broadcast* de todos os atributos do nó emissor e a filtragem dos mesmos pelos receptores.

O papel do programador baseia-se apenas na definição das vizinhanças de cada nó e das restantes interacções que constituem a aplicação. Uma vizinhança é definida por uma lista de vizinhos e pelo conjunto de atributos partilhados. O objectivo da solução é fornecer uma interface de acesso aos nomes dos vizinhos e seus atributos partilhados, com uma descoberta e troca de dados transparente para o utilizador.

O ponto chave do Hood é o mecanismo de *broadcast/filter*, empregue para suportar a difusão de atributos partilhados e a escolha desses mesmos pelos nós receptores. Desta forma, é implementada uma assimetria no seu funcionamento, onde os nós proprietários dos atributos enviados não possuem consciência dos nós receptores interessados nos valores difundidos e da sua captura. Tal como noutras soluções de *middleware*, este mecanismo permite uma comunicação assíncrona e desacoplada entre processos.

2.6.3 Abstract Regions

Com o objectivo de simplificar a concepção de aplicações para RSSF, o Abstract Regions [56] fornece um conjunto de primitivas de programação que abstraem mecanismos de baixo nível subjacentes, tais como comunicação, partilha de dados e operações colectivas. É baseada no conceito de região, que define uma relação de vizinhança entre um nó e outros nós da rede segundo certas propriedades, tais como conectividade ou localização geográfica.

De modo a esconder da aplicação os detalhes da disseminação e agregação de dados, esta solução é composta por operadores espaciais que capturam interacções entre as regiões. Tais operadores permitem a descoberta de vizinhos, a partilha, agregação e redução de dados, e a listagem de participantes de uma região. Os nós pertencentes a uma região podem distar por múltiplos *hops*, e a sua partilha de dados é suportada por um modelo de espaços de tuplos.

Adicionalmente, o Abstract Regions possibilita o balanço entre a fiabilidade e os recursos aplicados das comunicações, permitindo a adaptação face às condições dinâmicas da rede através do ajuste do consumo de energia e da largura de banda do meio de comunicação. Desta forma, é dada à aplicação o controlo dos recursos empregues nas comunicações e o acesso à informação sobre a precisão e completude das operações colectivas.

2.6.4 Context Shadow

No âmbito da computação ubíqua, o Context Shadow [28] é uma solução de organização e pesquisa de serviços, permitindo determinar o contexto de um utilizador e quais os serviços relevantes para esse contexto. O Context Shadow permite obter serviços pertinentes para um dado contexto, com base nas suas propriedades, como a localização (proximidade) ou relação organizacional. Permite também a estruturação dos serviços, com informação de contexto associada, num repositório para posteriores consultas.

No Context Shadow, algumas entidades são representadas como servidores de contexto, cujos actuam como repositórios da informação de contexto das mesmas. Essas entidades submetem informação no seu servidor correspondente, e ligam-se uns aos outros num modelo de comunicação com capacidades de bases de dados suportada por um espaço de tuplos. A informação contextual também é armazenada e transmitida pelos espaços de tuplos, suportando o dinamismo topológico.

Destacam-se três mecanismos chave no Context Shadow: **Referenciação Cruzada**, para ligações entre servidores de contexto, suportando o dinamismo da topologia da rede formada por essas entidades, **Descoberta de Recursos e Serviços**, proporciona a produção de *queries* usando o TSpaces ou XQL (XML Query Language) e a descoberta de serviços adaptada à arquitectura Jini, e **Refinamento da Informação de Contexto**, que, através de um Refinador de Contexto, permite a verificação e validação da informação submetida nos servidores de contexto.

2.6.5 Agilla

O Agilla [21] é uma plataforma de *middleware* para RSSF baseada em agentes móveis, que permite a injecção e migração dos mesmos pela rede. O propósito desta solução é suportar a execução de aplicações adaptativas ao dinamismo do ambiente circundante à rede. Em cada nó, é mantido um espaço de tuplos local, através dos quais os agentes se coordenam mediante operações locais ou remotas sobre esses espaços.

Sendo a abordagem de dinamismo baseada em agentes, é fornecida a possibilidade de se empregar migração fraca ou forte, consoante se deseje, respectivamente, uma transferência do apenas do código, reiniciando o estado da execução; ou uma transferência do código e estado para continuação da execução no próximo nó. O modelo do Agilla suporta a execução de quatro agentes em cada nó, aspecto que limita o desempenho face à escala. As mudanças de contexto são geridas de forma automática devido à autonomia dos agentes, e os espaços de tuplos privados possibilitam a sua execução

concorrente totalmente separada (sem acederem ao mesmo espaço).

Outro aspecto particular orientado ao dinamismo é o suporte a reacções dos agentes aquando das correspondências de tuplos a *templates* aplicados a espaços. Tal é importante na medida em que se evitam operações síncronas de polling. Finalmente, o Agilla pressupõe que o *addressing* a cada nó se baseia na sua localização geográfica, assumindo a sua determinação prévia através de um esquema de localização.

2.6.6 TeenyLime

O TeenyLIME [18] é um *middleware* direccionado para arquitecturas descentralizadas, com foco no suporte de coordenação entre dispositivos. Apesar de ser uma evolução do LIME [43], são mantidas a distribuição do espaço de tuplos pelos dispositivos, a partilha transitória de tuplos entre nós conforme conectividade, e operações reactivas relativas a tuplos de interesse. No entanto, abarca a diferença de restringir as partilhas transitórias de tuplos somente entre vizinhos directos (*one-hop*), onde cada nó possui uma visão parcial e diferente do espaço de tuplos.

É dada a possibilidade de restrição do escopo da operação ao espaço local, a um espaço existente num vizinho directo, ao espaço formado por todos os vizinhos directos ou ao espaço formado pelos ultimos mais o espaço local. Permite também operações *reliable* ou *unreliable*, que permite suportar maior, ou menor, fiabilidade na comunicação e coordenação, conferindo-lhe flexibilidade.

A importância da temporalidade dos dados é realçada no TeenyLIME com o uso de um mecanismo de frescura. O tempo é dividido em épocas de duração igual, e na criação de um tuplo, é-lhe associado um *timestamp* da época em que é criado. O mecanismo permite associar um *timestamp* máximo a um template, útil para limitar a frescura dos resultados; ou obter-se a frescura de um dado tuplo. Um dos pontos chave e das principais inovações introduzidas pelo TeenyLIME são os tuplos de capacidade (*Capability Tuples*), que permitem indicar aos participantes da rede que um dado nó possui a capacidade de produzir dados correspondentes a um dado *pattern*, paradigma converso ao modelo reactivo.

2.6.7 Discussão Sumária

O Hood é um paradigma pró-activo cujo mecanismo de *broadcast/filter* confere leveza computacional aos emissores. No entanto, peca por não ser reactivo e por usar apenas o padrão de comunicação *many-to-one*. Outro ponto negativo é a falta de controlo no *broadcast*, efectuado automática e periodicamente, não permitindo explorar *tradeoffs* nas operações de comunicação.

O Abstract Regions é uma abordagem algo limitada, visto permitir apenas operações de leitura e escrita síncronas nos pares *key-value*. O facto de não existirem mecanismos de notificação do surgimento de valores no sistema também contribui como limitação. Finalmente, apesar de agrupar nós em regiões segundo propriedades de rede passíveis de serem definidas, cada região requer uma implementação dedicada, sendo essa exigência cara.

A plataforma de computação ubíqua Context Shadow prima pela orientação à organização e pesquisa de serviços, permitindo determinar o contexto de um utilizador e os serviços relevantes para esse contexto, conforme propriedades dos serviços. Todavia, essa manutenção e fornecimento de serviços exige uma maior complexidade ao nível da aplicação. Outro ponto negativo é a falta de transparência na partilha de dados, pois requer o acesso explícito ao espaço de tuplos de interesse para se obterem os dados pretendidos.

Num plano diferente às restantes aproximações de *middleware* apresentadas nesta secção, o Agilla baseia-se em agentes móveis com execução autónoma e transparente. O Agilla é também composto por mecanismos de reacções a correspondências de *patterns* a tuplos. No entanto, apesar do seu objectivo ser retirar esforço e complexidade ao programador, os agentes são produzidos em código Assembly, o que é contraditório. A falta de transparência é outro ponto negativo, pois o espaço de tuplos geral é fisicamente distribuído mas não é visto como único. Finalmente, requer conhecimento sobre a localização do dispositivo, informação que exige técnicas pouco praticáveis ou eficientes em termos de consumo de energia.

Sendo o LIME uma abordagem de *middleware* direccionada para redes ad-hoc, serve como base para apresentação dos aspectos base do TeenyLIME. Este, é composto por um modelo de programação geral com métodos pró-activos e reactivos, e mecanismos orientados para RSSF, tais como comunicação *hop-by-hop*, *templates* com intervalos de valores associados e *Capability Tuples*, que permitem endereçar aspectos específicos como a gestão de energia. Os dados são representados por tuplos existentes em espaços partilhados de forma transitória e vistos como um único espaço de memória. A flexibilidade do TeenyLIME é elevada pois permite aplicar paradigmas de comunicação além do *many-to-one*, tais como *one-to-many* e *many-to-many*. Destaca-se a possibilidade de explorar o balanço entre fiabilidade e consumo de recursos das operações, consoante a sua semântica. Finalmente, é um sistema adaptado ao modelo de programação event-based do TinyOS e a sua implementação em NesC é pública [5].

2.7 Análise do trabalho relacionado e objectivos

Após a apresentação e análise crítica de abordagens relacionadas com o âmbito desta dissertação, pretende-se nesta subsecção formar a pilha estruturada da arquitectura objectivo. A pilha na figura 2.2 serve de referência para a concretização do *Middleware* de VLS que se concretizou neste trabalho, descrita no próximos capítulos.

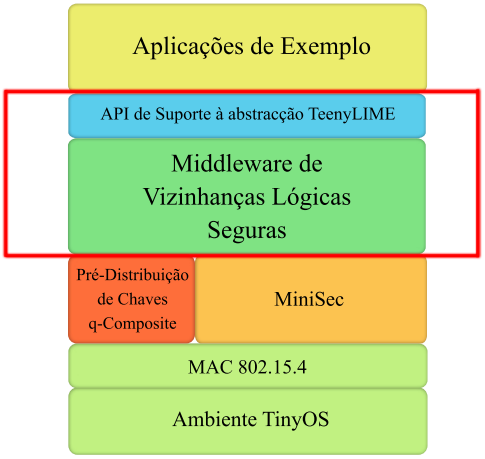


Figura 2.2: Estruturação de Abordagens dos Serviços de Segurança em Redes de Sensores sem Fios e respectivo foco de Implementação deste trabalho denotado a vermelho.

Das soluções de comunicação segura da secção 2.2, o MiniSec é a ideal visto obter elevada segurança com um consumo energético eficiente, suportando dois modos seguros de comunicação: ponto-a-ponto e multi-ponto. Por outro lado, o TinySec é uma aproximação simples e suficiente para se obter comunicação segura.

A análise da secção 2.3.1 conclui que o SecLEACH é um protocolo orientado para distribuição de chaves, pouco interessante para esta dissertação. O INSENS e o Clean Slate são abordagens mais relevantes, visto contemplarem encaminhamento seguro face a intrusões.

Nos serviços de gestão e estabelecimento de chaves da secção 2.4, o q-Composite é preferível por ser o mais resiliente, nas situações de intrusões de pequena escala.

Dos serviços de pesquisa e difusão da secção 2.5, o Secure Diffusion é o mais adequado, visto tornar seguro o Directed Diffusion, aproveitando os seus mecanismos de resiliência e adicionar técnicas de evitação e repulsa de intrusos.

Dos suportes de programação da secção 2.6, o TeenyLIME é o mais adequado visto a sua arquitectura de espaços de tuplos transitoriamente partilhados se adaptar melhor às RSSF, para além de ser flexível nos modelos de comunicação e garantir alguma fiabilidade na execução das operações disponibilizadas às aplicações.

3

Modelo e Arquitectura do Sistema de Vizinhanças Lógicas Seguras

Este capítulo apresenta formalmente a visão conceptual de uma arquitectura de referência da pilha de suporte às vizinhanças lógicas seguras para RSSF, concretizada perante os objectivos da presente dissertação. Esta pilha fornece, ao programador de aplicações para RSSF, uma API dividida em dois grupos de funcionalidades: operações sobre espaços de tuplos e funções de gestão e reorganização de vizinhanças lógicas. Internamente, o sistema é constituído por dois componentes principais: um serviço de gestão de espaços de tuplos e um serviço de pesquisa e difusão de dados. A escolha dos sistemas direccionados para cada um desses componentes é suportado pelo estudo realizado no capítulo anterior de Trabalho Relacionado. As seguintes secções expõem as funcionalidades e os níveis que constituem a pilha estruturada e a sua interligação lógica, pelo que o presente capítulo organiza-se da seguinte maneira:

- Primeiro, é apresentada na secção 3.1 a importância essencial do sistema de VLS proposto na presente dissertação, introduzindo o paradigma funcional oferecido pela arquitectura, e quais os aspectos contributivos trazidos pela utilização da mesma.
- A secção 3.2 abrange uma descrição introdutória do modelo de utilização do sistema das VLS, nomeadamente ao nível dos pressupostos da visão e acesso à RSSF que cada nó possui, as funcionalidades utilitárias oferecidas e quais as relações e precedências entre as operações.
- Na secção 3.3 é apresentada a visão conceptual da arquitectura, numa forma de estruturação em múltiplos níveis, apresentando os objectivos práticos de cada um dos mesmos, bem como a dependência processual e funcional entre si.
- A secção 3.4 materializa o modelo conceptual do sistema de vizinhanças lógicas

seguras, definindo quais os sistemas, dos existentes na investigação recente, que foram empregues na edificação da pilha de serviços implementada, bem como o seu processamento e elementos base necessários, e também a sua relação ao longo dos níveis estruturados.

- A secção 3.5 aborda a constituição e método de operação dos componentes principais da arquitectura do sistema de vizinhanças lógicas:
  - A subsecção 3.5.1 apresenta uma API de utilização da pilha de serviços implementada, dividida em dois grandes grupos de funcionalidades: operações sobre espaços de tuplos partilhados e métodos de reconfiguração e reorganização de vizinhanças lógicas.
  - A subsecção 3.5.2 descreve a constituição do serviço de espaços de tuplos partilhados incluído na implementação do sistema das vizinhanças lógicas, nomeadamente ao nível das operações que disponibiliza às camadas superiores e à forma como concretiza o paradigma.
  - A subsecção 3.5.3 ilustra o serviço de pesquisa e difusão de dados para RSSF aplicado no sistema das VLS. Este serviço serve como base ao acesso à rede visto pela camada de suporte a espaços de tuplos.
  - A subsecção 3.5.4 é dedicada ao enlace entre os dois serviços principais: o sistema de gestão de espaços de tuplos e o sistema de pesquisa e difusão de dados, acrescentando o componente necessário para concretização da gestão e organização de VLS.
- A secção 3.6 aborda a temática dos componentes de segurança incluídos na pilha implementada, não só ao nível das comunicações e protecção face a ataques externos, bem como os mecanismos de resiliência pró-activa face a ataques por intrusão.
- Na secção 3.7 é apresentado o conjunto de funcionalidades que a pilha de VLS concebida neste trabalho oferece, indo ao encontro de possíveis requisitos aplicacionais.
- Finalmente, a secção 3.8 ilustra as linhas gerais de desenho de alguns exemplos de aplicações que a arquitectura concebida suporta, fazendo uso das funcionalidades principais que a mesma oferece.

3.1 Foco da Arquitectura Concebida

Pretende-se, com a aplicação da arquitectura proposta, manter-se a expressividade de utilização dos sistemas similares de gestão de espaços de tuplos, acrescentando-se funcionalidades de descoberta e agregação/cobertura de vários nós capturadores

de eventos, possivelmente distantes entre si, bem como operações de gestão e manipulação dos grupos estabelecidos, suportados por um sistema de pesquisa e difusão de dados que é caracterizado por oferecer níveis interessantes de resiliência, escala e consumo energético.

Na arquitectura proposta na presente dissertação, a colocação do suporte de pesquisa e difusão de dados resulta no incremento do escopo das operações sobre espaços de tuplos face ao das concretizadas nos sistemas presentes da actual investigação de RSSF, nomeadamente o TeenyLIME, abordado na secção 2.6. A cobertura dos métodos baseia-se no conceito de VLS, introduzido no capítulo 1, estabelecida a partir de uma condição lógica definida ao nível aplicacional, transparecendo os saltos entre vizinhanças físicas directas que se interpõem entre os participantes de cada agrupamento e o originador da vizinhança. O sistema de pesquisa e difusão de dados aqui abordado, no qual a arquitectura das VLS é baseada, permite a um nó de acesso à rede efectuar a colecção de dados, em cada agrupamento lógico que crie, numa metodologia orientada à obtenção de eventos emitidos por múltiplos nós capturadores.

Outro ponto central que caracteriza a arquitectura recai sobre a adaptibilidade activa face ao dinamismo da rede, propriedade essa que se pretende que seja suportada por um mecanismo interno automático e transparente para o utilizador, concretizado ao nível do sistema de pesquisa e difusão de dados incorporado. Repare-se que o dinamismo da rede não só abarca a própria qualidade das ligações entre os elementos da mesma, como tem em vista a repulsa de nós que sofreram ataques por intrusão, isto é, que demonstrem um comportamento incorrecto e irregular.

Finalmente, são tidos em conta os pressupostos de segurança ao nível das comunicações, especificamente a confidencialidade, integridade e autenticidade de dados transmitidos, através da utilização de um módulo auxiliar ao nível da infra-estrutura base onde assenta a camada de MW das VLS.

3.2 Modelo de Utilização da Arquitectura

A incorporação da arquitectura das VLS tem como alvo uma rede de sensores típica ao estilo do modelo salientado na secção 1.2 do capítulo introdutório do presente documento. As linhas caracterizadoras das redes objectivadas são aqui descritas ao nível do papel de cada nó, sendo que os pormenores técnicos dos dispositivos e da implementação dos componentes *software* serão abordados no capítulo 4.

Cada elemento constituínte de uma RSSF que utilize o sistema das VLS possui o papel de nó sink ou de nó comum (ou ordinário), e, por sua vez, este segundo tipo engloba nós sensores (obtentores de eventos) ou actuadores (que alteram o seu meio

ambiente circundante físico). É requirido então que um ou mais elementos da rede permitam o acesso à RSSF por parte de estações-base, e que os restantes nós sejam alcançados por esses nós sink. Relativamente à aplicação, é possível que várias versões da mesma sejam executadas, mas apenas uma única das mesmas esteja activa em cada nó.

A figura 3.1 ilustra o modelo de utilização do sistema de VLS, nomeadamente ao nível da linha de execução das operações disponibilizadas às aplicações. Inicialmente, pretende-se que o utilizador defina os parâmetros necessários de criação e activação (início) de uma VLS, para só depois aplicar as operações sobre espaços de tuplos partilhados de forma compatível com os sistemas que implementam esse mesmo paradigma. Para além desse conjunto de operações, também os métodos de mudança de estado de uma VLS só podem ser aplicados após o início da mesma. Note-se que, a qualquer altura do tempo de vida de uma VLS, é possível aceder ao seu estado - tal característica é útil se, numa dada altura, uma aplicação pretender obter informação associada a uma VLS, como por exemplo, obter os identificadores de todos os participantes da mesma.

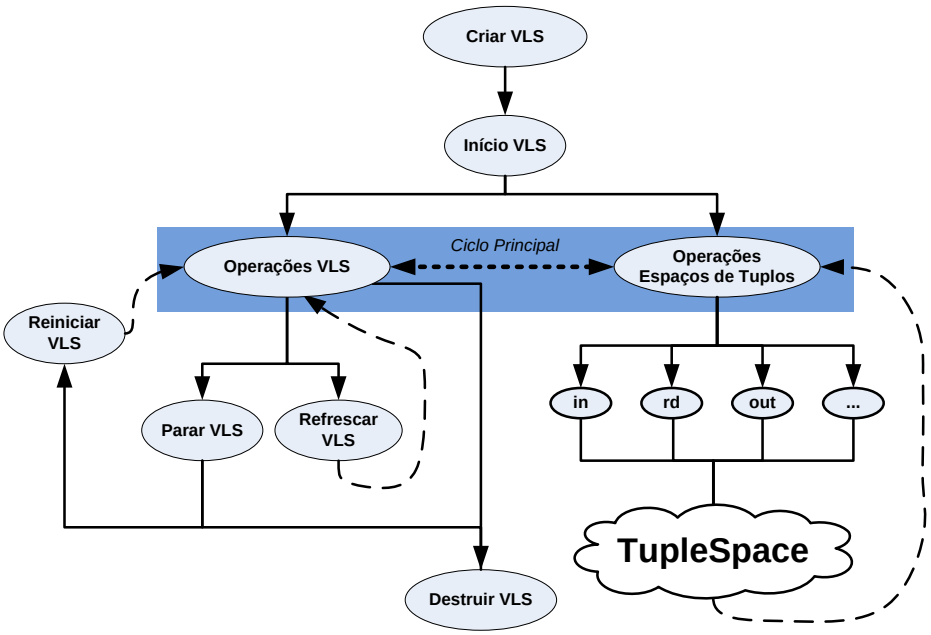


Figura 3.1: Modelo funcional de utilização do Sistema de VLS.

Durante a existência de uma VLS, pretende-se disponibilizar a oportunidade de suspensão da mesma, e futuro reinício, para possibilitar ao programador um adiamento, de longo termo, das operações. Finalmente, ao se efectuar a operação de destruição de uma VLS, está-se a forçar o fim de vida de um dado agrupamento lógico, pelo que todos os seus participantes serão notificados. Repare-se que uma VLS possui

associado um limite temporal de expiração do agrupamento, pelo que se esse valor é alcançado ou superado, é efectuada uma acção autónoma de desinscrição da VLS por parte de cada nó.

Outra utilidade do sistema de VLS passa por permitir a reorganização destes agrupamentos realizados numa RSSF, restruturação essa que se centra em conceitos de parâmetros temporais, como a frequência de actualização dos espaços de tuplos, ou para inclusão forçada de novos participantes numa VLS. No entanto, note-se que o próprio sistema de pesquisa e difusão de dados já concretiza uma descoberta de novos nós que serão agregados à tarefa correspondente despoletada no nó sink, como se indica na secção 3.5.3.

3.3 Arquitectura Conceptual

Com base nos objectivos desta dissertação, o modelo conceptual da pilha de suporte às vizinhanças lógicas seguras é estruturado nos diversos níveis representados na figura 3.2. Partindo deste modelo da arquitectura de referência, descrevem-se de seguida cada um dos níveis que compõem a pilha implementada:

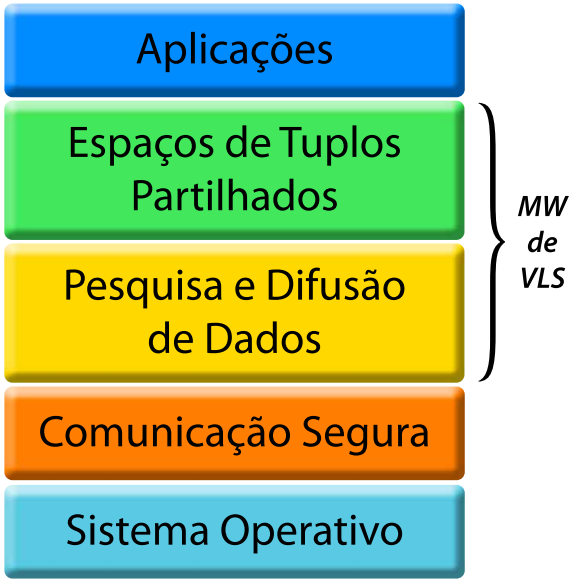


Figura 3.2: Arquitectura conceptual da pilha estruturada implementada, com realce do foco da implementação situado no *Middleware* das VLS.

- **Sistema Operativo:** Este nível representa a infra-estrutura base da arquitectura definida. É composta pelo sistema operativo, que fornece um suporte de compilação e build de aplicações para RSSF e, para além de outras funcionalidades

básicas como editores de ficheiros de código fonte ou "shell scripting", disponibiliza um ambiente de simulação e emulação real. Assim, esta porção da pilha de serviços deve ter em conta requisitos de baixo custo em termos de recursos, como capacidade de processamento, de memória e de consumo energético.

- **Comunicação Segura:** Apesar de, conceptualmente, este nível não ser parte integrante da infra-estrutura adoptada para a implementação da pilha das VLS, o seu acoplamento com a camada inferior é elevado tendo em conta que, geralmente, as operações de comunicação de baixo nível são disponibilizadas directamente pelo sistema operativo. Pretende-se então clarificar que este nível, fazendo também parte do suporte base, distingue-se da camada abaixo por possuir funcionalidades adicionais e pontuais de segurança, nomeadamente ao nível da comunicação ponto-a-ponto, conferindo as propriedades de confidencialidade, autenticidade e integridade de dados.
- **Pesquisa e Difusão de Dados:** Sobre o suporte base de implementação é colocado o sistema que fornece os serviços de pesquisa e difusão de dados em RSSF. Surgem neste nível os componentes que: fazem uso das primitivas de comunicação fornecidas pela camada inferior; realizam a gestão das rotas preferenciais para, consequentemente, providenciar aspectos de resiliência e adaptabilidade à RSSF, e que disponibilizam à camada superior primitivas de pesquisa e difusão de dados.
- **Espaços de Tuplos Partilhados:** É neste nível que o cerne da implementação se encontra, de forma a se elevar o paradigma das vizinhanças lógicas de um nível físico para um nível lógico. Neste processo de expansão estão envolvidas a concepção, especificação, implementação e validação dos serviços, de nível sessão, que estabelecem a abstracção das VLS como agregações de nós que partilham espaços de tuplos com uma escala de múltiplos *hops*. A segurança destes espaços de tuplos assenta no pressuposto de que existe uma garantia implícita das propriedades de segurança objectivadas a partir das camadas inferiores da pilha. Este nível fundamenta o corolário da implementação do sistema das VLS, disponibilizando um ambiente de programação e concretizando uma API com o mesmo tipo de expressividade subjacente a outras implementações do paradigma dos espaços de tuplos partilhados.
- **Aplicações:** Aqui situam-se os programas para os quais o sistema de VLS está direccionado. Pretende-se que as aplicações mantenham o mesmo conteúdo semântico relacionado com o paradigma dos espaços de tuplos partilhados, mantendo-se a coerência nos processos de *porting* entre a arquitectura desenvolvida nesta dissertação e outros sistemas que implementam esse mesmo paradigma. Adicio-

nalmente, as aplicações que operem sobre o sistema de VLS têm à sua disposição métodos adicionais, relativos à restruturação e reorganização destes agrupamentos lógicos.

3.4 Arquitectura Instanciada

Pretende-se com esta secção apresentar, de forma concreta, quais os componentes que formam os vários níveis do sistema implementado, reservando um maior enfoque no cerne da implementação: o nível *middleware* das VLS. A figura 3.3 ilustra cada um desses componentes principais, e sua relação funcional, para cada nível da pilha implementada, com base na arquitectura conceptual de referência descrita acima na secção 3.3.

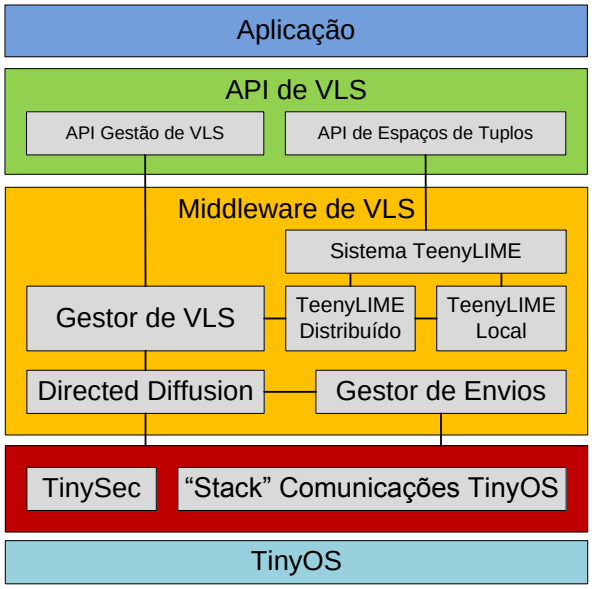


Figura 3.3: Arquitectura instanciada e componentes principais da pilha estruturada das Vizinhanças Lógicas Seguras.

A API do sistema de VLS concretizado nesta dissertação divide-se em dois componentes: API de Gestão de VLS e API de Suporte a Espaços de Tuplos. A primeira interface é suportada pelo sistema Gestor de VLS que, passe-se a redundância, suporta a gestão das mesmas, usando directamente o suporte de pesquisa e difusão para alcançar os possíveis participantes de um agrupamento lógico. Por outro lado, a segunda interface da API principal é suportada pelo sistema de gestão de espaços de tuplos partilhados concretizado pelo Sistema TeenyLIME, abordado no capítulo 2.6.6 da investigação recente das RSSF, acedendo a um ponto cimeiro do MW. Esta estratificação de pontos de acesso ao MW conferem à API das VLS um acesso de estilo multinível,

possibilitando à aplicação um teor de flexibilidade que vai desde criar e aceder a informação de uma VLS até criar uma VLS e gerir o espaço de tuplos partilhado da mesma.

Relativamente ao restante suporte a espaço de tuplos partilhados, o Sistema TeenyLIME é o seu ponto de acesso, concretizando a comutação operações locais e distribuidas, suportadas, correspondentemente, pelo componente TeenyLIME Local e pelo TeenyLIME Distribuído. Conceptualmente, esta organização é idêntica à do *middleware* TeenyLIME original, mantendo-se as seguintes funcionalidades operadas sobre o repositório de tuplos local ou distribuído:

- Operações de inserção, leitura e remoção de tuplos do repositório;
- Instalação e desinstalação de reacções ao surgimento de tuplos no repositório.

A concepção do suporte a espaços de tuplos partilhados mantém-se inalterada face ao seu desenho geral, exceptuando no facto de ser necessário agregar-se às operações um conjunto de informação identificativa das VLS. Tal justifica-se pela necessidade de se elevar a noção de vizinhança concretizada pelo sistema original, que, na pilha de serviços das VLS, não é representada pela vizinhança física directa mas sim por outros dispositivos além desse conjunto de nós.

Com base no ponto conclusivo do parágrafo anterior, é necessário existir um mapeamento entre uma vizinhança criada e uma qualquer operação sobre o espaço de tuplos associado à mesma. Tal é conseguido pela obtenção de uma identificação única da VLS no momento da criação da mesma e o seu fornecimento por parâmetro em cada operação de espaço de tuplos efectuada. O sistema de vizinhanças incluído no TeenyLIME acaba por ser desprezado, sendo agora essa tarefa incumbida ao gestor de VLS, por motivos directamente relacionados com sua funcionalidade de criação e atribuição de identificadores às vizinhanças formadas.

Além de gerir as vizinhanças às quais o nó actual está associado, o gestor de VLS mantém informação adicional sobre os agrupamentos lógicos nele despoletados, como o tempo actual até à sua expiração. Isso permite verificar se qualquer operação sobre o espaço de tuplos da VLS pode ser efectuada ou não, e com base nessa necessidade que surge a relação entre o Gestor de VLS e o TeenyLIME Distribuído.

O Suporte à Pesquisa e Difusão de Dados serve como forma de acesso à RSSF, incorporando os mecanismos de disseminação e de resiliência de fluxo de dados, com base nas linhas conceptuais do Directed Diffusion, sistema esse que foi também apresentado na secção 2.5.2 do trabalho relacionado. Em termos dos objectivos funcionais da arquitectura de VLS, este seu sistema interno de pesquisa e difusão de dados suporta:

- Pesquisa de nós que correspondam a uma VLS;
- Devolução dos eventos capturados por participantes de uma VLS;

- Operações de gestão de VLS, como inicio, destruicao, suspensao e reinicio de VLS;
- Disseminação pelas rotas criadas das operaçoes distribuídas sobre espaços de tuplos partilhados;
- Resposta às operações efectuadas pelas rotas;
- Modificação da frequência da obtenção de dados de uma VLS;
- Automatização da inserção e remoção de participantes de uma VLS.

Ao nível das comunicações, a pilha de VLS incorpora um componente intermediário para gerir as mensagens a enviar. A necessidade da existência deste módulo surge com a problemática da frequência excessivamente elevada dos envios de mensagens realizados por um nó da RSSF, que pode, eventualmente, gerar um número significativo de tentativas falhadas. Essa falha deriva do facto do módulo de envio poder já estar a ser utilizado para enviar uma outra mensagem anterior, pelo que a transmissão de uma nova só incorre em sucesso se a que se encontrava a ser transmitida tiver sido completamente enviada. Assim, de forma a não se descartarem mensagens de saída que não podem ser enviadas no momento, estas são entregues ao Gestor de Envios.

O componente interno de gestão de envios, contido na arquitectura de VLS, manipula um *buffer* circular de mensagens pendentes, para mais tarde serem enviadas sem qualquer tipo de interrupção. Este módulo baseia-se na pilha de comunicação do TinyOS para realizar as transmissões. Relativamente à recepção de mensagens, é também usada a mesma pilha de primitivas de comunicação, mas directamente ligada ao suporte de pesquisa e difusão de dados, que trata as mensagens recebidas por um nó. Esta pilha de comunicação é complementada pelo módulo TinySec, que confere à arquitectura as propriedades de segurança ao nível das comunicações: autenticação, confidencialidade e integridade de dados, e que será abordado na secção 3.6.

Na seguinte secção 3.5 descreve-se, de forma mais refinada, cada um destes componentes que formam todo o *middleware* das VLS desenvolvido na presente dissertação.

3.5 Componentes do Sistema de Vizinhanças Lógicas

Como funcionalidade principal do sistema de VLS surge a disponibilização de operações sobre espaços de tuplos partilhados, concretizados sobre agrupamentos de nós baseados em requisitos lógicos definidos ao nível aplicacional. Para tal, é efectuado um acoplamento entre um sistema de gestão de espaços de tuplos para RSSF e um sistema de pesquisa e difusão de dados. De uma forma geral, foi incorporado ao paradigma das vizinhanças lógicas o conceito de que o escopo de um espaço de tuplos partilhados pode alcançar nós além da vizinhança física de um dado nó. O sistema de VLS é assim

composto por 3 componentes centrais: Suporte de Espaços de Tuplos, Gestor de VLS e Sistema de Pesquisa e Difusão de dados; os quais são descritos nesta secção.

3.5.1 Métodos Disponibilizados pela Arquitectura

Segundo os requisitos definidos pelos objectivos a alcançar pelo sistema das VLS, este é utilizado por meio de uma API que combina dois grupos de operações: suporte a métodos sobre espaços de tuplos e gestão de vizinhanças lógicas seguras.

Primeiramente, pretende-se fornecer um conjunto de operações sobre espaços de tuplos partilhados, ao estilo do sistema TeenyLIME, de forma a se manter a expressividade de programação face às aplicações já desenvolvidas para esse mesmo sistema. A tabela 3.1 ilustra este conjunto de operações sobre espaços de tuplos, dividido em operações relacionadas com tuplos singulares, múltiplos tuplos, reacções ao surgimento de tuplos e tuplos específicos.

Operaçoes sobre Espaços de Tuplos Partilhados			
Tuplos Singulares	Grupos de Tuplos	Reacções a Tuplos	Tuplos Específicos[1]

`out`
`rd`
`in`
	`rdg`
	`ing`
		`addReaction`
		`removeReaction`
			`reifyNeighborTuple`
			`reifyCapabilityTuple`

1 - As operações relativas a tuplos específicos ao TeenyLIME não incidem sobre os espaços de tuplos mas são disponibilizadas na API de forma a se obter acesso a esses elementos.

Tabela 3.1: Tabela ilustrativa do conjunto de operações sobre espaços de tuplos partilhados a disponibilizar pelo sistema das VLS.

No conjunto das operações relativas a tuplos singulares, o método `out` permite a um nó escrever um tuplo no espaço de tuplos, o `rd` retorna uma cópia de um tuplo mantendo o original, e o `in` retira um tuplo do espaço de tuplos para um nó. Para as duas últimas operações de acesso ao espaço de tuplos devolverem um tuplo, é empregue um mecanismo de endereçamento associativo, concretizado através da passagem por parâmetro de um tuplo com alguns campos por definir - denominado *template* ou *pattern* - e encontrando um tuplo, no espaço de tuplos, cuja correspondência de atributos é válida. O método de comparação entre esses elementos será descrita, de forma detalhada, na secção 4.4.4.

Em termos das operações de grupos de tuplos, as operações `rdg` e `ing` possuem o mesmo efeito que, correspondentemente, os métodos `rd` e `in`, com a diferença de que são retornados todos os tuplos no espaço de tuplos que fazem correspondência com o *template* fornecido. A importância destes métodos mais abrangentes remete para a

obtenção ou remoção completa de uma dada informação existente no repositório de dados.

As primitivas de `addReaction` e `removeReaction` permitem instalar uma reacção a um tuplo sobre um espaço de tuplos. As reacções efectivam o objectivo de quando um tuplo surja no repositório, e corresponda ao *template* associado à reacção adicionada, esse mesmo tuplo seja retornado ao nó que a instalou. O uso destes elementos remete a um estilo de operações de subscrição de tuplos, permitindo o adiamento de execução até os dados surgirem.

As restantes operações `reifyNeighborTuple` e `reifyCapabilityTuple` são orientadas, correspondentemente, aos tuplos de vizinhança e aos tuplos de capacidade. São disponibilizadas aos módulos aplicacionais, que utilizarem a API, para terem acesso a esses tipos especiais de tuplos, que são bastante específicos ao modelo do TeenyLIME, mantendo-se assim a expressividade oferecida por esse sistema. Assim, com o método `reifyNeighborTuple`, permite-se que a aplicação local tenha acesso a um tuplo que o identifica perante os seus vizinhos. No entanto, o sistema de gestão de vizinhanças agora é concretizado de uma forma central ao nó que criou uma VLS, e não como é realizado no TeenyLIME, onde cada nó da RSSF mantém um conjunto de identificadores da sua vizinhança directa [18]. O acesso aos tuplos de capacidade, fornecido pela operação `reifyCapabilityTuple`, é igualmente oferecido na API das VLS, devido à importância da abstracção de associação de um conjunto de operações (código) aquando do surgimento de um tuplo numa vizinhança, suportada por esses mesmos elementos.

Operaçoes sobre Vizinhanças Lógicas			
Construção de VLS	Actividade de VLS	Refrescamento de VLS	Obtenção de Informação
createVLS destroyVLS	startVLS restartVLS stopVLS	refreshVLS	getInfoVLS

Tabela 3.2: Tabela ilustrativa do conjunto de operações sobre vizinhanças lógicas a disponibilizar pelo sistema das VLS.

Adicionalmente, o segundo grupo de operações disponibilizado pela API das VLS pretende disponibilizar primitivas de definição, alteração de estado ou actividade, refrescamento segundo novos parâmetros e obtenção de informação e participantes de uma VLS. Este grupo de métodos está listado na tabela 3.2.

Os métodos de definição de VLS permitem realizar o agrupamento de nós dada a informação necessária para tal processo, como a duração máxima da sua actividade, o intervalo com que se recebe informação dos elementos que a compõem e as condi-

ções de definição dos nós participantes da mesma. Em termos concretos ao primeiro método, a VLS irá extinguir-se após a duração definida ter expirado, mas é disponibilizada a operação de terminação de uma VLS, pelo que os nós participantes serão forçados a abandonar a VLS.

São também disponibilizadas operações sobre VLS activas, listadas na segunda coluna da tabela 3.2. A operação `startVLS` tem como pré-condição a VLS já ter sido criada no momento do seu uso, mediante a primitiva `createVLS`. Só após a execução da operação de início de uma VLS é que é possível terminar a actividade da mesma, através do método de `stopVLS`. Estando a VLS num estado inactivo, para voltar a colocá-la em operação, utiliza-se o método `restartVLS`. Estas três operações são bastante interessantes na medida em que permitem à aplicação suspender o estado de uma VLS, para mais tarde a execução desse agrupamento ser novamente despoletada.

Com a operação `refreshVLS` pretende-se disponibilizar uma forma de restruturação de uma VLS a qualquer momento da sua existência. Para tal, podem ser fornecidos novos dados associados a um agrupamento lógico, como uma nova duração ou frequência de obtenção de tuplos, mas, no limite, o objectivo desta funcionalidade abarca a reorganização dos elementos constituintes de uma VLS, permitindo a adição, no momento, de novos nós ao agrupamento lógico, caso estes existam. Repare-se que esta primitiva não permite a remoção forçada de nós, pois, tal como apresentado nas contribuições da arquitectura proposta, esse é um processo automático intrínseco ao suporte de pesquisa e difusão de dados que a pilha de serviços inclui.

3.5.2 Suporte de Espaços de Tuplos Partilhados

Como referido anteriormente, o sistema de suporte à programação sobre espaços de tuplos partilhados é um dos componentes principais do MW de VLS. Este componente disponibiliza à aplicação as operações a efectuar sobre os repositórios de tuplos, por meio de uma API própria para o efeito, que contém as operações indicadas na anterior tabela 3.2. A sua funcionalidade é idêntica à disponibilizada por um dos sistemas apresentados no anterior capítulo 2 de trabalho relacionado, o TeenyLIME, que se concluiu ser o mais adequado perante os objectivos e contribuições da presente dissertação.

Em termos gerais, o TeenyLIME baseia-se no objectivo de agrupar nós sensores e actuadores nos mesmos grupos, numa metodologia baseada na sua proximidade física. Estes agrupamentos de nós permitem a coordenação dos dispositivos mediante uma abstracção suportada por espaços de tuplos partilhados de forma transitiva. Este conceito de espaço de tuplos federado é concretizado pela partilha de dados apenas com os vizinhos directos. Face aos restantes nós, cada um deles possui assim uma visão diferente do espaço de tuplos transitivo, visto possuir diferentes vizinhos dos

que os outros têm. No caso do suporte de espaços de tuplos concretizado na presente dissertação, a limitação do escopo dos agrupamentos de nós pode ser superior à vizinhança física directa, pelo que a visão do espaço de tuplos por parte dos participantes do mesmo agrupamento lógico é a mesma. Cada participante de uma VLS possui a mesma visão do espaço de tuplos de cada agrupamento lógico a que pertença, visto esse repositório ser centralizado no nó sink, originador da vizinhança. Esta gestão e coordenação de espaços de tuplos distribuídos será abordada, com maior detalhe, na secção 3.5.4.3.

Uma característica importante inerente ao sistema das VLS, presente no suporte a espaços de tuplos partilhados, abarca os vários paradigmas de comunicação possíveis. Para além do típico *many-to-one*, caracterizado pelos múltiplos fluxos proactivos de dados em direcção a um nó, os paradigmas *one-to-many* e *many-to-many* são explorados por este serviço de difusão, concretizados pela abilidade de definição de vários nós da RSSF como destino dos dados. As operações de colecção de tuplos, após o estabelecimento de uma VLS, são caracterizadas como *many-to-one*, onde os vários participantes de uma VLS capturam e emitem dados, direccionados para o *sink* através do sistema de difusão. O paradigma *one-to-many* está presente, não só no processo de criação de uma VLS, como no aplicação de operações sobre os espaços de tuplos dos participantes de uma VLS. Por fim, a metodologia de comunicação *many-to-many* é vísivel nos cenários em que uma VLS é criada por múltiplos nós *sink*, mas que não são abordados no presente trabalho e que se deixam como trabalho futuro.

O sistema produzido como componente de gestão de espaços de tuplos da pilha de VLS partilha do mesmo modelo arquitectural do TeenyLIME, e é composto por 3 componentes internos: a interface TeenyLIME, sistema LocalTeenyLIME e sistema DistributedTeenyLIME. O primeiro componente TeenyLIME serve como ponto de partida da utilização deste sistema, e tem como principal funcionalidade a comutação entre as operações sobre espaços de tuplos com escopo local, implementadas no sistema LocalTeenyLIME, e as com escopo remoto (i. e. abarcando a vizinhança), suportadas pelo DistributedTeenyLIME. Esta divisão é realizada, tal como no sistema TeenyLIME original, devido à importância da independência entre a gestão dos tuplos locais e o processamento distribuído [18]. A descrição destes dois componentes é apresentada de seguida.

3.5.2.1 Componente de Espaço de Tuplos Local

O sistema de espaço de tuplos local suporta os métodos realizados sobre o repositório de dados tipificados do nó onde se encontra presentemente inserido, oferecendo igualmente uma gestão e armazenamento desses mesmos dados. Dos métodos ofere-

cidos pela interface de espaços de tuplos, descrita na tabela 3.1, toda a vertente local dessas operações - *out*, *rd*, *in*, *rdg*, *ing*, e adicionalmente, *addReaction* e *removeReaction* - é implementada neste componente, bem como os processos de correspondência de tuplos.

São assim oferecidas as operações: *out(t)*, que insere no repositório local o tuplo t suplementado por argumento; *rd(p)*, que permite a leitura de um tuplo, oriundo do espaço de tuplos local, cuja correspondência com a *pattern* p fornecida seja válida; *in(p)*, que, tal como a operação de leitura, permite obter do espaço de tuplos local um tuplo cuja correspondência com a *pattern* p fornecida seja válida, mas removendo esse mesmo tuplo; *rdg(p)*, que realiza a leitura de todos os tuplos, existentes no repositório local, e que tenham correspondência com a *pattern* fornecida como argumento; e finalmente, *ing(p)*, que permite remover do espaço de tuplos local todos os tuplos com os quais a *pattern* p faça correspondência.

A gestão e armazenamento do repositório local de tuplos é realizada neste componente tendo em conta as operações efectuadas neste espaço de tuplos, bem como a frescura da informação, associada a cada um desses elementos tipificados. Para além de manter a informação individualmente contida em cada elemento da RSSF, este componente concretiza a gestão de reacções ao surgimento de tuplos neste mesmo repositório. Uma reacção, após ser instalada por este componente, irá reagir a uma operação de inserção de um tuplo no espaço de tuplos do nó onde foi instalada. Este processo de disparo de uma reacção realiza-se logo após cada colocação de um tuplo no repositório local, pelo que se tenta verificar a correspondência entre o tuplo colocado e a *pattern* associada à reacção.

3.5.2.2 Componente de Espaço de Tuplos Distribuído

O componente de espaço de tuplos distribuído oferece à aplicação os métodos da tabela 3.1, nas quais o seu escopo de operação é externo. Como referido anteriormente no início do presente capítulo, um dos requisitos do sistema é permitir a realização de operações sobre o espaço de tuplos partilhado com os participantes de uma vizinhança, sendo que para tal, este componente possui como base um componente de comunicação, conceptualmente suportado pelo sistema de pesquisa e difusão, descrito de seguida em 3.5.3. Desta forma, o componente de espaço de tuplos distribuído serve de intermediário entre um nó e os restantes participantes de uma VLS.

As operações distribuídas sobre espaços de tuplos partilhados são efectuadas num processo de comunicação de uma mensagem, contendo um tuplo e a operação correspondente, enviada para todos os participantes de uma vizinhança e processada pelos destinatários. Este processamento de uma mensagem recebida é também realizado

pelo componente de espaço de tuplos distribuído, tratando assim as operaçoes despoletadas pelo emissor da mensagem, manipulando assim o seu espaço de tuplos local. Assim, outro papel do sistema de espaço de tuplos distribuído é realizar a ligação das operações efectuadas por outros elementos da RSSF sobre o repositório local do nó receptor, suportada por uma interface de ligação:

BridgeTupleSpace: A interface de BridgeTupleSpace possui um papel central no escalonamento das operações entre os vários participantes de uma VLS, permitindo uma ligação entre o componente de espaço de tuplos local e o distribuído. Após o despoletar de uma operação sobre outro nó da vizinhança, o destinatário recebe uma mensagem que será tratada pelo componente de espaço de tuplos distribuído, tornando-se necessária a disponibilização de operações de manipulação do repositório local de tuplos presente nesse participante. Após a realização de uma operação, a interface define também uma forma de resposta, para fornecer ao componente distribuído um meio de retorno do resultado da aplicação dessa operação sobre o espaço de tuplos local.

Assim, esta interface de BridgeTupleSpace serve como o suporte combinatório entre os intervenientes do processamento distribuído, permitindo a realização de operações sobre os participantes da vizinhança, bem como a produção do correspondente resultado. São oferecidas as mesmas operações que a API das primitivas sobre espaços de tuplos locais, mas com a diferença de ser necessário distinguir-se se o tuplo retornado é resultado do disparo de uma reacção ou de uma operação sobre espaços de tuplos.

3.5.3 Suporte de Pesquisa e Difusão de Dados

Como suporte à pesquisa e difusão de dados, o serviço implementado é baseado no Directed Diffusion. Este sistema, para além de possuir características de robustez e resiliência no seu processo de execução devido à escolha adaptativa de rotas, é orientado à colecção de dados numa perspectiva *data centric*, realizando pesquisas por *named data*, e efectuando o caching, agregação ou descarte de informação [27].

A sua funcionalidade principal é obter um volume completo de informação que surge numa RSSF, coincidente com uma *query* previamente realizada pela aplicação, e de uma forma eficiente ao nível do fluxo de dados, do consumo energético e da escalabilidade da rede. O seu método de operação, descrito nos seguintes pontos desta secção, é composto pela disseminação de *queries* pela rede iniciada por um nó específico, pela consequente resposta dos nós que possuem dados coincidentes com as interrogações realizadas, e pelo aumento da taxa de obtenção de eventos por rotas preferenciais.

Neste protocolo de pesquisa e difusão de dados, é atribuído a cada nó um de três possíveis papéis: nó *sink*, que é considerado o ponto de acesso à rede e, a partir do qual, se disseminam *queries* de pesquisa sobre a mesma, e que recolhe todos os eventos de

resposta capturados e enviados pelos elementos da rede; nó *source*, que corresponde a um elemento da rede que possua informação que faça correspondência com a pesquisada, sendo assim um dos coleccionadores de eventos oriundos do meio envolvente à rede; nó intermediário, que intervém somente no processo de difusão de interesses, devolução de dados e reforços de rotas.

Este sistema é caracterizado por alcancar a propriedade de resiliência no processo de comunicação de informação, quer na fase de disseminação de interesses como na de colecção de dados, descritas nos pontos seguintes, através de dois mecanismos: formação de múltiplas rotas entre um *sink* e os vários sources, e reforços selectivos da qualidade de uma, ou mais, dessas rotas estabelecidas na primeira fase de disseminação de interrogações.

Difusão de Interesses: O processo de execução deste sistema inicia-se com a definição de uma tarefa de pesquisa, a qual recebe, para além da *query*, uma duração limite da mesma e uma frequência da recepção de dados. A *query* é necessária para se produzir o interesse correspondente à tarefa, que será difundido por toda a rede, numa metodologia *hop-by-hop*. Esse interesse inclui os restantes elementos temporais, que determinam durante quanto tempo, e com que frequência, os nós *source* difundem, de volta ao *sink*, os seus dados coincidentes com a pesquisa realizada.

Formação de Gradientes: Os "gradientes" são ligações representativas que indicam o sentido do fluxo de dados a partir de um dado nó. Estas ligações são formadas aquando da difusão dos interesses, onde um nó cria um gradiente na direcção do nó que lhe enviou um interesse. Pretende-se assim que, para cada interesse existente na cache de um nó, estejam associados a essa entrada os vários gradientes formados com o mesmo, isto de forma a que, posteriormente, seja possível direccionar dados ao longo da rede para o nó *sink*. A cada gradiente está também associado um valor de frequência, que caracteriza a qualidade do *link*, e que será alterado conforme os reforços aplicados ao longo da rede, e, consequentemente, nessa mesma ligação.

Devolução de Dados: No processo de difusão dos interesses, figurativos da pesquisa despoletada, cada nó consulta a sua cache de dados e, caso possua uma entrada que faça correspondência com esse interesse, inicia uma tarefa de envio da informação de resposta à pesquisa. Essa tarefa possui uma frequência de envio definida pelo valor de intervalo contido no interesse, realizando um envio directo dos dados para os gradientes associados ao mesmo. Adicionalmente, a sua tarefa de colecção de eventos decorre até um tempo limite, igualmente especificado pela duração do interesse correspondente. O envio dos dados capturados pelos sources seguem múltiplas rotas, definidas ao longo de saltos mediante os gradientes formados.

Reforço de Rotas: Na recepção de dados oriundos dos nós *source*, o sistema de difu-

são despoletado no *sink* reforça um ou vários caminhos entre si e a origem dos dados, isto é, aumenta a frequência de obtenção de dados de uma ou mais rotas preferenciais. O processo de reforço é concretizado pelo envio de interesses, com um valor de intervalo menor, de forma directa para o nó cuja ligação com o qual pertença à rota preferencial. Este processo é repetido por quem recebe o reforço, até que o mesmo alcance o extremo final da rota se pretende reforçar, e, consequentemente, o nó *source* aumente a frequência de envio de dados. A tarefa de amostragem que já havia sido delegada a um *source* que receba um reforço mantém-se, mas agora com os parâmetros de intervalo e duração contidos no mesmo, que serão diferentes do interesse original. Este processo permite, adicionalmente, a degradação de caminhos menos preferenciais, devido a uma diminuição de qualidade dos *links*, através envio de reforços negativos, que, basicamente, correspondem a interesses com valor de intervalo mais elevado, para resultar numa menor frequência de obtenção de eventos pelos mesmos.

Na sua forma original, a aproximação do Directed Diffusion, que serve de referência para a implementação do serviço de pesquisa e difusão de dados da presente arquitectura, baseia-se, em grande parte, numa metodologia *Many-to-One*, caracterizada pelos vários fluxos de dados formados entre os nós *source* e o *sink*, após a fase de disseminação de interesses e formação de gradientes. O paradigma *One-to-Many* também está presente, sendo representado pela fase inicial de pesquisa. O processo de reforço de rotas preferenciais é caracterizado por um modelo de comunicação *One-to-one*, onde um nó melhora a taxa de obtenção de eventos oriundos de um outro nó singular.

3.5.4 Suporte de Gestão de Vizinhanças Lógicas Seguras

Os sistemas de espaços de tuplos e de pesquisa e difusão de dados são combinados por meio de um componente que realiza a gestão das VLS, como é ilustrado na figura 3.3 da arquitectura instanciada. Este gestor permite o enlace entre estes dois sistemas principais na medida em que o suporte de pesquisa e difusão de dados é usado como a ferramenta de obtenção dos participantes de uma VLS, servindo igualmente como o motor para a posterior manipulação dos espaços de tuplos locais desses participantes.

De uma forma geral, é concretizado um mapeamento entre o escopo das operações sobre espaços de tuplos com os *targets*, isto é, com quem se partilha o repositório de agregação lógica. A aplicação, ao invocar a primitiva de estabelecimento de uma VLS - implementada por este mesmo componente -, irá obter o identificador desse agrupamento lógico para que, ao realizar posteriores primitivas sobre espaços de tuplos distribuídos, possa indicar em que VLS as mesmas irão ser efectuadas.

3.5.4.1 Relação com os Componentes Principais

Este componente usa directamente, do Suporte de Pesquisa e Difusão de Dados, as operações de início de uma difusão (quer seja para criar, suspender ou terminar uma vls) ou para induzir um reforço de uma VLS (isto é, envio de reforços positivos). É também realizada a manipulação das mensagens recebidas por cada nó que utilize a pilha de VLS, implementando desta forma o evento de recepcao de dados do sistema de pesquisa e difusão, para que em cada recepção de informação sobre uma VLS mantida pelo nó em questão, esta informação seja gerida.

Por outro lado, o componente de gestão de VLS é utilizado pelo sistema de gestão de espaços de tuplos para suportar as operações sobre espaços de tuplos distribuídos, isto é, operações sobre as VLS. Através do identificador de VLS, obtido pela aplicação, o componente de espaços de tuplos distribuídos indica qual a VLS a que uma operação corresponde, enviando assim para os participantes da RSSF associados a esse identificador. Todos os resultados de uma operação de espaços de tuplos sobre uma vizinhança são também intermediados por este componente gestor, derivados da diferenciação entre mensagem de gestão de VLS ou de espaços de tuplos distribuídos.

Adicionalmente, para além de realizar o suporte do mapeamento entre participantes de uma VLS, este sistema de gestão permite à aplicação um uso quase que directo do suporte de pesquisa e difusão de dados, possibilitando a simples funcionalidade de obtenção da informação associada à VLS, tal como a obtenção dos nós que correspondam a um dado conjunto de condições que formem um agrupamento lógico.

3.5.4.2 Alterações ao Modelo Original dos Sistemas Principais

Por forma a se conseguir combinar os dois sistemas principais da arquitectura das VLS (Suporte a Espaços de Tuplos e Suporte a Pesquisa e Difusão de Dados), é necessário acrescentar informação relativa à vizinhança com que uma dada operação esteja relacionada, quer ao nível da gestão de espaços de tuplos, como ao nível da gestão dos agrupamentos lógicos. Ao nível do suporte de pesquisa e difusão de dados, o início de uma tarefa de obtenção de dados está associada à criação de uma vizinhança lógica, pelo que é necessário, tanto os interesses, como os dados, possuirem a informação que identifica sobre qual VLS se está a realizar a operação.

Ainda relativamente ao sistema de difusão da pilha de VLS, é acrescentada a funcionalidade de envio directo de mensagens para os *sources* associados a uma tarefa de obtenção de eventos. A metodologia segue as linhas de operação do mecanismo de reforços, ao nível do tipo de comunicação empregue: é enviada uma mensagem especial para todos os emissores de dados de forma endereçada, e não por difusão completa, aproveitando todos os caminhos já formados.

No caso do sistema de gestão de espaços de tuplos, apenas é necessário que este consiga indicar para qual VLS uma operação de processamento distribuído se destina, identificador esse obtido por argumento externo fornecido pela aplicação. De resto, os métodos de gestão de espaços de tuplos mantêm-se inalterados.

3.5.4.3 Gestão e Coordenação Distribuída de Espaços de Tuplos

Contrariamente ao modelo do TeenyLIME, os espaços de tuplos partilhados na arquitectura proposta são mantidos, na totalidade, nos nós *sink*, e resultam da combinação dos espaços de tuplos locais de cada participante da VLS. Esta propriedade é suportada pela característica principal do sistema das VLS, a de agrupamento de participantes, por parte de um dado nó, e colecção de dados oriundos dos mesmos. Desta forma, cada participante de uma VLS possui a mesma visão do espaço de tuplos correspondente ao agrupamento lógico a que pertence, exceptuando os espaços locais dos restantes participantes da VLS. Este particionamento em espaços de tuplos locais é adequado às RSSF, permitindo que cada nó opere sobre o seu espaço de tuplos sem existir uma dependência de coordenação directa com os restantes participantes.

A manipulação de espaços de tuplos remotos no sistema das VLS é iniciada a partir do momento em que um nó, ao receber o respectivo pedido de pesquisa por participantes lógicos, consequentemente se qualifique como participante da mesma. Na recepção desse pedido, a linha de operação sobre esse espaço de tuplos encontra-se ao nível aplicacional, conferindo flexibilidade e controlo por parte das aplicações, ao nível pró-activo, face a criação de VLS. Assim, cada participante opera sobre o espaço de tuplos de uma VLS a que pertença, remetendo qualquer operação ao nó *sink* originador desse agrupamento.

Posteriormente, após qualquer alteração do espaço de tuplos de uma VLS, quer por parte de uma operação remota realizada por um participante da mesma, quer por parte do próprio *sink*, este nó, que é o originador da VLS, mantém o espaço de tuplos numa metodologia de centralização: qualquer nó participante acede ao sink para manipular o espaço de tuplos correspondente a VLS em questão, e qualquer operação realizada pelo *sink* é enviada para todos os elementos da rede, na forma de reforços directos do sistema de difusão.

Apesar deste modelo de centralização de tuplos - onde os dados de uma VLS convergem apenas para o criador da mesma - poder não ser, à primeira vista, adequado para RSSF, pois podem-se originar pontos de contenção da rede nos nós *sink* em situações de um número elevado de VLS activas, o protótipo não contempla esses cenários. No entanto, assume-se que este problema não surgirá se as VLS forem criadas por diferentes elementos da rede, uniformizando, de certa forma, essa carga de processamento

e armazenamento.

3.6 Segurança do Sistema de Vizinhanças Lógicas Seguras

A arquitectura proposta na presente dissertação contempla a manutenção de propriedades de segurança ao longo dos vários níveis da pilha, desde a sua infra-estrutura base até ao nível cimeiro do MW. A aplicação pode assim assumir a garantia dessas mesmas propriedades, oferecidas pelo sistema de VLS implementado na presente dissertação. Assumindo o modelo de estruturação de serviços em RSSF inicialmente apresentado na figura 1.1, a segurança ao nível aplicacional irá depender das camadas *middleware* e sistema operativo, pelo que a presente secção se divide nos mecanismos de segurança ao nível: das comunicações face a ataques externos; e do processo de disseminação de dados face a ataques internos.

3.6.1 Segurança das Comunicações

O modelo arquitectural das VLS assume as propriedades básicas de segurança garantidas por módulos do sistema operativo onde a pilha MW assenta, colocando a responsabilidade da segurança das comunicações sobre um nível proximamente relacionado com o sistema operativo. Essas propriedades obtidas pela arquitectura dependem assim do sistema empregue para tornar as comunicações seguras, mas assumem-se que as propriedades básicas, nomeadamente confidencialidade, autenticação e integridade dos dados, são garantidas pelo módulo TinySec. Segundo [32,40], este sistema de segurança de camada *link* consegue ultimar essas propriedades básicas, obtendo um impacto leve na performance e memória. A descrição dos mecanismos que compõem este sistema foram apresentados anteriormente na secção 2.2.2.3, e, acrescenta-se que a integração deste sistema na arquitectura do MW das VLS é directa, pois faz parte integrante do sistema operativo TinyOS como um módulo de utilização opcional.

3.6.2 Resiliência a Ataques Externos

A capacidade de resiliência à acção de ataques externos da pilha das VLS é obtida a partir do sistema de difusão e pesquisa de dados aplicado na estruturação do sistema desenvolvido na presente dissertação. Este sistema oferece dois mecanismos que conferem resistência ao processamento distribuído numa RSSF: formação e utilização de múltiplas rotas, e reforço selectivo de rotas preferenciais.

O mecanismo de formação de múltiplas rotas, concretizado ao nível do sistema de difusão de dados, permite que toda, ou grande parte, da RSSF seja coberta pela pesquisa, conseguida pelo processo de disseminação total de interesses. Após a formação das múltiplas rotas entre *sink* e *sources*, o mecanismo de reforço selectivo de rotas opera de forma a que seja escolhido o caminho considerado preferencial, quer ao nível da qualidade dos dados, como ao nível do comportamento dos elementos da rede.

Apesar do Directed Diffusion não ter em conta a garantia de níveis de segurança, dos ataques apresentados na secção 2.3.1 do trabalho relacionado, consegue-se com os mecanismos de multi-rota e de reforços selectivos, atenuar os ataques por descarte de mensagens. No entanto, na presença de uma forma de detecção de intrusões, o protocolo pode basear o mecanismo de reforços ao nível, não só da qualidade dos dados, mas também ao nível da sua autenticidade. A aproximação de segurança do Directed Diffusion, apresentada na secção 2.5.4 do trabalho relacionado, consegue torná-lo num protocolo suficientemente seguro, mas tal implementação não foi possível na presente dissertação devido ao elevado esforço necessário para o concretizar no ambiente de desenvolvimento da mesma.

3.7 Funcionalidades do Sistema

As principais funcionalidades oferecidas pelo sistema de VLS correspondem, de uma forma geral, às primitivas disponibilizadas pelo *middleware* de VLS proposto, apresentadas na anterior secção 3.5.1. Nesta secção descrevem-se as possibilidades funcionais que essas operações oferecem no processo de desenvolvimento de aplicações direccionadas ao paradigma das VLS:

- É possível estabelecer uma VLS composta por todos os nós de uma RSSF que possuam um tuplo que corresponda a um dado *template*, representativo da condição lógica a partir da qual se constrói o agrupamento de participantes;
- Após o início de uma VLS, é possível verificarem-se quais os nós da RSSF que a ela estão associados. O MW de VLS pode utilizar e transportar essa informação ao longo da rede, até ao *sink*, independentemente da forma do esquema de identificação, desde que a informação de identificação esteja presente em cada nó;
- São, obviamente, oferecidas as funcionalidades de gestão de espaços de tuplos, agora correspondentes a VLS, para aplicações desenvolvidas em torno desse paradigma. É apenas necessário que uma aplicação explicite qual a vizinhança que pretende como destino de uma operação distribuída;

- Disponibiliza-se uma forma de aumentar ou diminuir a frequência de obtenção de eventos de uma VLS, através do reforço de rotas entre o *sink* e os participantes da VLS, permitindo que o sistema se adapte a uma possível variabilidade da necessidade de obtenção de dados por parte das aplicações;
- Oferece-se a utilidade de suspensão e posterior reinício de uma VLS, orientada para requisitos de aplicações com processamento muito esporádico, onde os períodos de tempo entre obtenções de eventos são significativamente longos.

3.8 Exemplos de Possíveis Aplicações

Primeiro, e remetendo novamente às contribuições da presente dissertação, pretende-se que a pilha de implementação das VLS suporte aplicações que mantenham grande parte da sintaxe e semântica das desenvolvidas para outros sistemas do mesmo paradigma. De forma mais específica, não é necessário realizarem-se grandes alterações nas aplicações desenvolvidas para o sistema TeenyLIME, para que estas funcionem correctamente no sistema de VLS.

A adaptação de aplicações é, de forma geral, simples, passando pela definição da condição lógica de formação da vizinhança, e, a partir daí, realizar-se a linha de operações sobre espaços de tuplos concretizada na aplicação original. É também necessário ter-se em conta o aumento do escopo das operações, que pode não especificar uma linha de execução exactamente igual à da aplicação original, e cuja vantagem tem de ser tida em conta no desenvolvimento de uma aplicação para as VLS.

As aplicações perspectivadas para utilização da pilha de VLS baseiam-se em ciclos de: realização de um agrupamento lógico de um conjunto de dispositivos e a posterior colecção de eventos recolhidos pelos mesmos. De forma mais sofisticada, este modelo aplicacional pode ser extendido numa aproximação de múltiplas VLS, onde um nó *sink* pode simultaneamente criar múltiplos agrupamentos lógicos e obter eventos de ambos, realizando, por exemplo, operações de conjuntos sobre esses mesmos, tais como a união ou intersecção de VLS.

Um primeiro exemplo de aplicação que o sistema das VLS suporta passa pela simples descoberta dos elementos da RSSF que correspondam a uma dada condição, permitindo ir ao encontro da problemática referente a um típico estilo de questões: "se existem e quais os nós que possuem a capacidade de obter um certo tipo de eventos?". Este tipo de aplicações faz uso directo de um nível inferior da camada de MW das VLS, onde o suporte de gestão de espaço de tuplos não é utilizado no processo de despoletação de uma pesquisa de participantes do agrupamento lógico. A aplicação fornece ao processo de pesquisa o *pattern* relativo à condição de estabelecimento da VLS, sendo

que cada elemento da rede que possua um tuplo que origine uma correspondência, responde ao nó *sink* com dados que incluem esse mesmo tuplo.

Outro exemplo de aplicações utilizadas sobre a pilha de VLS passam pela manipulação de espaços de tuplos após a criação de uma VLS. Este é o grupo de aplicações desenvolvidas até agora para sistemas similares como o TeenyLIME. Por exemplo, a aplicação de prevenção de incêndios usando espaços de tuplos, descrita em [6], poderia igualmente ser implementada com o sistema de VLS: é criada uma vizinhança contendo os sensores de temperatura e outra contendo os sensores de fumo, ao invés de se instalarem reacções em todos os sensores por parte dos actuadores.

Um exemplo mais sofisticado que os anteriores está relacionado com a combinação de múltiplas VLS, descrito anteriormente nesta secção. Um *sink* pode criar duas, ou mais, vizinhanças, e, posteriormente, realizar leituras de tuplos de todas, podendo operar numa metodologia baseada em teoria de conjuntos. Um exemplo de uma aplicação de detecção de incêndios passaria por se criar uma VLS com todos os nós que sejam detectores de fumo. Na ocorrência de um tuplo que corresponda a um evento da presença de fumo, podem-se descartar eventos de interferência de outras origens que não a pretendida. Tal é conseguido pela criação de uma VLS adicional contendo nós que capturem eventos de temperatura superior a um determinado valor (relacionado com incêndios), e realizando a intersecção dos conjuntos de dados obtidos com as duas vizinhanças. Assim, é possível suportar aplicações que requiram maior precisão e refinação dos eventos capturados.

O desenvolvimento de programas pode evoluir para a inclusão de um maior suporte à gestão de VLS, sendo que com o sistema proposto nesta dissertação, é oferecido um escopo superior à vizinhança física directa. Para além da obtenção de informação sobre participantes actuais de um agrupamento lógico em especial, é possível uma aplicação realizar operações de gestão de VLS. Destaca-se a possibilidade da operação de refrescamento de uma vizinhança, com o objectivo de se alterarem intervalos de captura de eventos ou despoletar uma reorganização da rede, podendo incluir nesse agrupamento lógico novos participantes descobertos no momento deste processo.

Finalmente, as aplicações podem obter um nível de complexidade elevado, combinando todas as linhas das aplicações descritas acima, com um processamento dividido em épocas muito longas, com uma obtenção de eventos curta e irregular. Um exemplo de aplicação passa pela monitorização de uma área, realizada uma vez por dia, podendo ser interrompida por longos períodos de tempo, fazendo uso das funcionalidades de suspensão e posterior reinício de uma VLS.

4

Implementação da Arquitectura

Este capítulo contém os detalhes do processo de implementação do sistema aqui realizado. Primeiramente, a secção 4.1 apresenta uma descrição do ambiente de desenvolvimento e simulação, bem como das ferramentas auxiliares de visualização da execução de testes e de medição de gastos energéticos. A organização de RSSF para a qual a implementação das VLS está dirigida, é contemplada na secção 4.2. Depois, em 4.3, é descrito o módulo de segurança empregue para ultimar as propriedades básicas de segurança objectivadas. De seguida, na secção 4.4, realiza-se um enfoque no desenvolvimento dos componentes principais do sistema de vizinhanças lógicas seguras, nomeadamente: no *middleware* de espaços de tuplos, no sistema de pesquisa e difusão de dados e no sistema de gestão e manutenção de vizinhanças lógicas, e a sua interligação. Finalmente, é descrito em 4.5 o processo de medição do consumo energético, formando uma das bases de avaliação da pilha concebida.

4.1 Ambiente de Desenvolvimento e Simulação

Foi adoptada uma infra-estrutura básica de RSSF com plataforma Mica [34] e suporte TinyOS versão 1.x, utilizando como ambiente de simulação o TOSSIM [36, 37]. De entre as várias versões deste sistema operativo de redes de sensores, foi escolhida a versão 1.x, face às mais recentes 2.0.2 ou 2.1 - mais precisamente, a versão 1.0.1 -, devido à incompatibilidade entre os vários componentes da infra-estrutura para diferentes versões do TinyOS. A razão principal deve-se à falta de uma implementação do MiniSec para a plataforma Mica - apenas direccionado para Telos motes [38] -, impossibilitando a sua utilização sobre o emulador TOSSIM, tendo então de se empregar o módulo TinySec, disponível apenas na versão 1.x do TinyOS. Esta decisão recaiu também na afinidade do componente de visualização de simulação TinyViz com os

restantes componentes, nomeadamente com o PowerTossim [51], apenas presentes no TinyOS 1.x. A seguinte figura 4.1 apresenta uma visão global da implementação, indicando quais os componentes, *hardware* e *software*, nos quais a pilha das VLS se baseia.

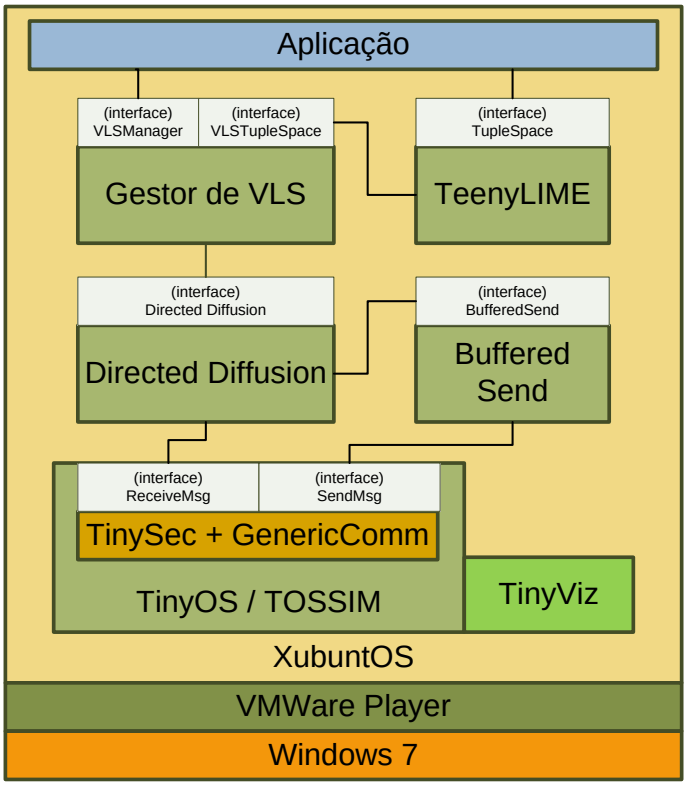


Figura 4.1: Visão global do ambiente de desenvolvimento e simulação, ilustrando a disposição dos principais componentes de plataforma base e do *middleware* das VLS, nos quais as aplicações se suportam.

A plataforma de hardware onde todo o desenvolvimento das soluções implementadas foi realizado, bem como o conjunto de testes efectuados para a validação experimental, é composta por um sistema que possui um processador Intel(R) Core(TM)2 Duo a 1.86GHz e 2GB de memória RAM a 666MHz de FSB. Nesta máquina encontra-se instalada uma versão 32bit do sistema operativo Windows 7, no qual corre uma instância da distribuição XubuntOS 2.1 sobre uma máquina virtual VMWare Player com a versão 2.5.2, instalado e configurado conforme descrito em [7, 8].

Tal como na implementação do paradigma das vizinhanças lógicas concretizado no sistema TeenyLIME, a linguagem nesC [9, 22] torna-se vantajosa no desenvolvimento do sistema proposto na presente dissertação. A principal razão para tal é a sua execução *split-phase*, realizando uma metodologia própria para ambientes *sense-and-react*, com programação face a reacções e adequando-se a implementações com reduzidos consumos de energia. Desta forma, foi utilizada a versão 1.1.3, com os devidos *pat-*

ches sugeridos em [8]. A versão do suporte instalado de linguagem Java corresponde à 1.5.0.

O ambiente de simulação possui como componente principal o emulador TOSSIM [36, 37], um simulador de eventos discretos para aplicações TinyOS, permitindo a depuração, teste e análise das mesmas num ambiente controlado e repetitivo. Este sistema foca-se na alta fidelidade, simulando a rede ao nível do bit, contemplando capturas ADC individuais e todas as interrupções. No entanto, não modela perfeitamente ambientes de mundo real, fornecendo antes abstracções de fenómenos reais, tais como erros de bit nas operações realizadas ou anotações de ocorrência das mesmas, produzindo assim informação sobre a qual programas externos de pós-processamento possam actuar. Repare-se também que o TOSSIM baseia-se em algumas assumpções simplificadas, as quais podem não corresponder com uma execução em sensores reais, como por exemplo, é assumido que as interrupções são não-preemptivas, enquanto que num dispositivo actual, uma interrupção pode disparar enquanto outro código está a ser executado [36].

O emulador TOSSIM constroi a simulação directamente do código TinyOS, aproveitando a vantagem de se manter inalterado o código produzido para sensores reais. Após o *build* da aplicação para simulação, mediante o comando `make pc`, o ficheiro TOSSIM executável é utilizado com o seguinte comando:

```
./build/pc/main.exe [flags] <n_nos>
```

onde se define o conjunto de opções por meio de flags - tais como, `-h` para ajuda, `-t=<sec>` para limite máximo temporal, ou `-p` para anotações de energia por operação - e o número de nós constituintes da RSSF a modelar.

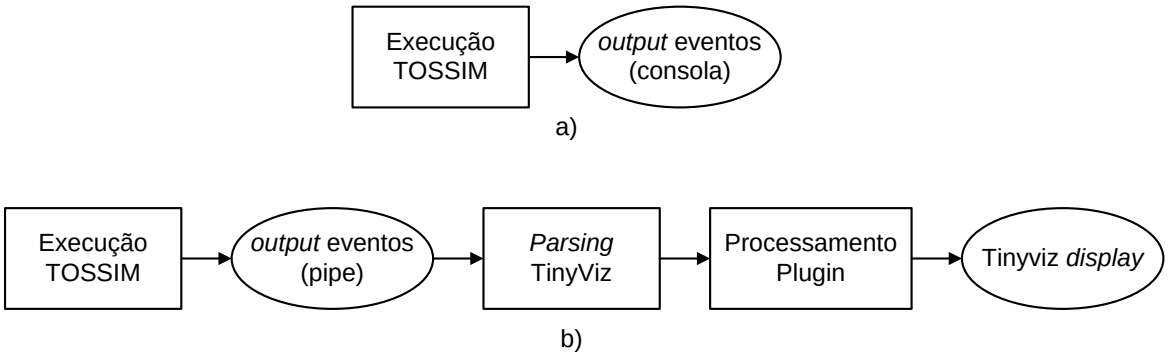
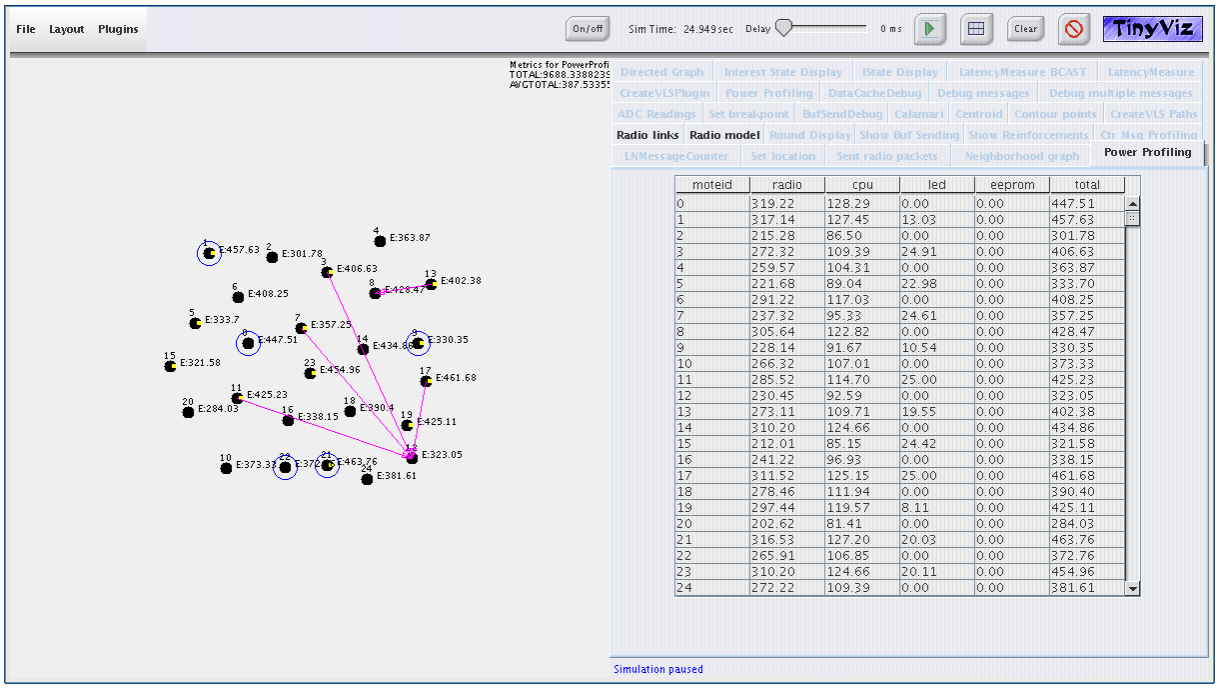


Figura 4.2: Modelo geral de execução de uma aplicação em TinyOS a) típico sem suporte gráfico ou b) com suporte gráfico TinyViz.

O módulo TinyViz [36] contido no package do TinyOS 1.0.1 possui como utilidade

a representação gráfica do decorrer da execução de um programa no ambiente TOSSIM. Tal é conseguido mediante uma implementação em linguagem Java que captura os eventos no *output* feito pelo próprio simulador, formalizando-os em elementos visuais numa interface gráfica. Através da execução numa consola do comando `tinyviz -run ./build/pc/main.exe <n_nos>`, surge uma janela como a ilustrada na figura 4.2, na qual se pode controlar e visualizar a simulação.

O TinyViz oferece a possibilidade interessante de se acrescentarem *plugins* simples, que permitem combinar eventos pontuais que o programador inseriu na aplicação - por meio de *prints* (primitiva `dbg()` do nesC) -, e reproduzidos em elementos gráficos, como ilustrado no esquema da figura 4.2. Tome-se como exemplo de *plugin* o PowerProfiling, ilustrado na figura 4.3, que é utilizado no presente trabalho como componente do modelo de energia, baseado no PowerTossim [52]. O seu método de operação basea-se na captura do *print* de cada operação efectuada por cada nó (envio ou recepção de rádio, ligar ou desligar led, etc.), realizando os necessários cálculos de gasto energético de cada uma, e, finalmente, na soma e apresentação ao utilizador do consumo total por tipo de evento ocorrido.



moteid	radio	cpu	led	eeprom	total
0	319.22	128.29	0.00	0.00	447.51
1	317.14	127.45	13.03	0.00	457.63
2	215.28	86.50	0.00	0.00	301.78
3	272.32	109.39	24.91	0.00	406.63
4	259.57	104.31	0.00	0.00	363.87
5	221.68	89.04	22.98	0.00	333.70
6	291.22	117.03	0.00	0.00	408.25
7	237.32	95.33	24.61	0.00	357.25
8	305.64	122.82	0.00	0.00	428.47
9	228.14	91.67	10.54	0.00	330.35
10	266.32	107.01	0.00	0.00	373.33
11	285.52	114.70	25.00	0.00	425.23
12	230.45	92.59	0.00	0.00	323.05
13	273.11	109.71	19.55	0.00	402.38
14	310.20	124.66	0.00	0.00	434.86
15	212.01	85.15	24.42	0.00	321.58
16	241.22	96.93	0.00	0.00	338.15
17	311.52	125.15	25.00	0.00	461.68
18	278.46	111.94	0.00	0.00	390.40
19	297.44	119.57	8.11	0.00	425.11
20	202.62	81.41	0.00	0.00	284.03
21	316.53	127.20	20.03	0.00	463.76
22	265.91	106.85	0.00	0.00	372.76
23	310.20	124.66	20.11	0.00	454.96
24	272.22	109.39	0.00	0.00	381.61

Figura 4.3: Captura de ecrã de uma execução TOSSIM com recurso ao visualizador TinyViz, ilustrando o plugin PowerProfiling.

4.2 Organização de Redes de Sensores sem Fios

A organização de um conjunto de elementos computacionais de uma RSSF é concretizada, de uma forma por defeito, pelo próprio TOSSIM. Após a chamada ao comando descrito na secção 4.1 de execução do emulador, este simula a colocação de nós numa área definido a sua posição num espaço de duas dimensões. Por defeito, a distribuição é aleatória sobre a área pretendida, e criando o raio máximo do alcance das comunicações para cada nó.

Com a utilização do TinyViz, tanto as posições dos nós como o seu raio de alcance podem ser redefinidas, sendo possível manter-se, de forma permanente, as localizações de cada elemento da rede, por meio de ficheiros `.mps` de topologia desse visualizador gráfico. Este módulo já possui formas de se posicionarem os nós de forma ordenada numa grelha, aleatória sobre toda a área ou aleatória em forma de grelha. O modelo de rádio que define o raio de alcance de cada dispositivo pertencente à rede pode ser igualmente redefinido, descartando também o modelo imposto pelo TOSSIM. Para tal, é utilizado um plugin chamado Radio Model, e é empregue em toda e qualquer simulação efectuada no presente trabalho, usando a mesma configuração que será indicada como parâmetro de testes experimentais no capítulo 5.

Uma das contribições da presente dissertação insere uma nova propriedade de auto-organização de RSSF, onde este processo baseia-se numa agregação de elementos de uma rede com base em condições lógicas. Assim, a organização de RSSF engloba o agrupamento de nós que possuam dados correspondentes a uma condição lógica definida ao nível da semântica da aplicação. Através de um sistema de espaços de tuplos, representativos da informação contida no sistema, cujo processamento distribuído é suportado por um sistema de pesquisa e difusão de dados, é dada aos nós de um agrupamento lógico a possibilidade de se coordenarem e de comunicarem eventos capturados, realizando uma organização por vizinhanças cujo escopo de acção vai além da vizinhança física directa.

4.3 Módulo de Comunicação Segura TinySec

O sistema de comunicação segura TinySec vem incluído na distribuição do TinyOS 1.x, sendo parte integrante deste sistema operativo como um módulo operacional para o TOSSIM, possuindo como plataforma alvo os Mica motes [32]. O seu requisito de instalação passa apenas por incluir na variável PATH do TinyOS o *script* `mote-key`, sendo que, na distribuição *standard*, tal já se encontra feito.

Em termos aplicacionais, é necessário realizar duas alterações directas: uma no Ma-

```
command result_t setTransmitMode(uint8_t mode)
command result_t setReceiveMode(uint8_t mode)
```

Tabela 4.1: Primitivas de definição do modo de operação do TinySec para casos de transmissão e de recepção.

kefile da aplicação e outra no código da mesma. Relativamente ao Makefile, basta incluir o Makerules existente na directoria tinyos-1.x/apps e, adicionalmente, a linha TINYSEC=true. O modo de utilização do TinySec é definido no código da aplicação, quando o componente TinySec é inicializado, através das primitivas NesC apresentadas na tabela 4.1. Como definido no manual em [31], para cada primitiva de definição do modo que se pretende que o TinySec opere, pode-se definir um dos três argumentos expostos na tabela 4.2.

setTransmitMode	setReceiveMode
TINYSEC_AUTH_ONLY*	TINYSEC_RECEIVE_AUTHENTICATED*
TINYSEC_ENCRYPT_AND_AUTH	TINYSEC_RECEIVE_CRC
TINYSEC_DISABLED	TINYSEC_RECEIVE_ANY

Tabela 4.2: Tabela ilustrativa dos possíveis modos do TinySec para, respectivamente, cada primitiva de transmissão e recepção. (* - Modos do TinyOS por defeito)

É interessante referir que a primitiva `setReceiveMode` estabelece, *a priori*, uma definição do nível de segurança das comunicações, pelo que o modo, definido por defeito, `TINYSEC_RECEIVE_AUTHENTICATED`, descarta mensagens de emissores em modo `TINYSEC_DISABLED` e aceita apenas dos outros dois. O valor `TINYSEC_RECEIVE_CRC` aceita mensagens somente com emissores em modo `TINYSEC_DISABLED`. Finalmente, o valor do modo de recepção `TINYSEC_RECEIVE_ANY` é mais flexível, aceitando transmissões de emissores em qualquer modo do TinySec.

Relativamente à chave utilizada pelo TinySec, esta é designada em tempo de *build* da aplicação. Na primeira vez que o TinySec é utilizado numa aplicação, é criado um ficheiro "key-file", denominado `tinyos-keyfile`, que contém uma chave por defeito e, possivelmente, outras chaves. Através da interface `TinySecControl`, oriunda do componente `TinySecC`, é possível renovar a chave a utilizar ou reiniciar o vector de inicialização.

4.4 Middleware de Vizinhanças Lógicas Seguras

O sistema de VLS tem como objectivo fornecer os métodos de gestão de agrupamentos lógicos e de espaços de tuplos partilhados que anteriormente foram descritos na secção 3.5.1, por meio da combinação do sistema Directed Diffusion com o *middleware* TeenyLIME, sendo que a sua API em linguagem nesC é apresentada na tabela 4.3.

4.4.1 API Disponibilizada

O conjunto de comandos e eventos definidos como interface do sistema de VLS encontra-se ilustrado na tabela 4.3, e vai de encontro aos dois conjuntos de métodos introduzidos em 3.5.1.

Operações sobre VLS - interface VLSManager

```
command result_t create(vlsOpId_t *opId, tuple *p);
command result_t destroy(vlsOpId_t *opId);
command result_t start(vlsOpId_t *opId);
command result_t stop(vlsOpId_t *opId);
command result_t restart(vlsOpId_t *opId);
command result_t refresh(vlsOpId_t *opId, tuple *p);
command result_t getInfo(vlsOpId_t *opId, vlsInfo_t *info);
```

Operações sobre Espaços de Tuplos - interface TupleSpace

```
command result_t out(TLOpId_t *opId, VLSIDT vlsId, tuple *t);
command result_t in(TLOpId_t *opId, VLSIDT vlsId, tuple *p);
command result_t rd(TLOpId_t *opId, VLSIDT vlsId, tuple *p);
command result_t ing(TLOpId_t *opId, VLSIDT vlsId, tuple *p);
command result_t rdg(TLOpId_t *opId, VLSIDT vlsId, tuple *p);
command result_t addReaction(TLOpId_t *opId, VLSIDT vlsId, tuple *p);
command result_t removeReaction(TLOpId_t *opId, TLOpId_t *reactionId);

event result_t tupleReady(TLOpId_t operationId, tuple *tuples, ...
                                                    ... uint8_t number);
event result_t reifyCapabilityTuple(tuple* capTuple);
event tuple* reifyNeighborTuple();
```

Tabela 4.3: Tabela ilustrativa da API fornecida pela pilha de Vizinhanças Lógicas Seguras.

Todas as primitivas `command` recebem uma estrutura de dados representativa da operação a realizar, podendo ser do tipo `TLOpId_t` ou `vlsOpId_t`, conforme se realize uma chamada ao sistema de Espaços de Tuplos ou ao sistema de Gestão de VLS. A aplicação fornece essa estrutura vazia e apenas inicializada, com o intuito do MW

a modificar para incluir o identificador e tipo de operação realizada. Posteriormente, em cada evento de recepção de um tuplo, a aplicação recebe igualmente uma estrutura de dados relativa à operação relacionada com esse tuplo, e pode, assim, comparar e determinar se alguma das suas chamadas tem como resultado esse tuplo recebido.

As operações sobre espaços de tuplos recebem agora, como argumento do escopo da operação, um elemento do tipo `VLSIDT`. O elemento identificador da VLS é obtido através da estrutura resultante da criação de uma VLS, `vlsOpId_t`, que contém informação sobre a mesma, incluíndo o identificador a fornecer nas operações sobre espaços de tuplos. No entanto, para se definir o escopo das operações sobre o espaço de tuplos local, foi reservado o valor 0 no argumento `vlsId`, pelo que o domínio númerico dos identificadores de VLS é iniciado pelo valor 1.

4.4.2 Elementos e Estruturas de Dados

A arquitectura concretizada neste trabalho engloba como elementos básicos os tuplos, formando a base da informação contida e utilizada na RSSF, assumindo duas formas: *patterns* ou tuplos. Os *patterns* representam um tipo de informação incompleta - com um ou mais campos formais -, que pode corresponder a dados completos, tendo assim o papel de suporte às *queries* efectuadas, sendo que os tuplos concretos representam a informação actual. A tabela 4.4 ilustra a constituição de um tuplo, e descreve cada campo do mesmo.

Os tuplos são criados pelo método `initTuple()`, interno ao sistema, onde se define o seu tipo, um conjunto de *flags* e se corresponde a um tuplo de capacidade ou não. De seguida, é necessário afectarem-se os campos que se pretendam que o tuplo contenha, mediante as primitivas específicas de afectação de campos formais, ou actuais: `setFormalUInt16Field` ou `setActualUInt16Field`, para um campo do tipo `unsigned int` de 16 bit. Existem igualmente outras primitivas para o caso de outros tipos de dados, tais como `unsigned int` de 8 bit, de 32 bit, ou caracteres. Cada campo do tipo `tlField` corresponde a um par tipo-valor que pode vir a ocupar 5 bytes [1], mas devido às limitações técnicas das RSSF, fixou-se um número máximo de 3 campos por tuplo.

Para se verificar a correspondência entre elementos de informação no sistema de VLS, ou seja, entre tuplos, é necessário estar implementado um algoritmo de correspondência de atributos dos campos que contêm. Segundo [27], o Directed Diffusion, sistema de referência para suporte a pesquisa e difusão de dados no MW de VLS, oferece a possibilidade de se aplicar livremente o método pretendido, característica

[1] Para o valor do campo de um tuplo, foi empregue o tipo de dados do nesC struct union, que contém um dos vários tipos de dados suportados e exigindo a memória correspondente ao mesmo.

`typedef struct tupleStruct {`	
`uint16_t logicalTime;`	tempo de criação
`uint16_t expireIn;`	limite de frescura
`uint8_t type;`	tipo de tuplo[1]
`bool isCapTuple;`	indica se é um tuplo de capacidade
`tlField fields[MAX_TUPLE_FIELDS];`	conjunto de campos tipo-valor do tuplo
`} tuple;`	

1 - Reserva-se o campo type para possíveis definições específicas da aplicação.

Tabela 4.4: Tabela representativa da constituição de um Tuplo.

mantida na arquitectura desenvolvida na presente dissertação.

O método de determinação da correspondência entre tuplos utilizado no sistema de VLS é o algoritmo de *one-way match* [24], caracterizado por se basear na comparação de atributos dos tuplos envolvidos. Neste algoritmo, os atributos formais de um tuplo são comparados com os atributos actuais de outro, ao estilo das linhas definidas para o paradigma de espaços de tuplos para a correspondência entre os mesmos [18, 23]. A seguinte figura 4.4 apresenta o pseudo-código do algoritmo de *one-way match*.

given two attribute sets *A* and *B*
for each attribute *a* in *A* **where** *a*.op is formal{
  matched = false
  for each attribute *b* in *B* **where** *a*.key == *b*.key **and** *b*.op is an actual
    if *a*.val compares with *b*.val using *a*.op, **then** matched = true
  if not matched **then** return false (no match)
}
return true (successful one-way match)

Figura 4.4: Algoritmo de correspondência de atributos One-Way (retirado de [24]).

4.4.3 Ciclo de Sistema e Temporalidade

O ciclo de simulação do sistema rege-se num *timer*, repetitivo e sem fim, que dispara num período de 500 ms, correspondendo assim a um limite teórico de reacção do MW, caracterizando a capacidade de resposta (*responsiveness*) do sistema. Este ciclo incide sobre ambos os subsistemas que constituem o serviço de VLS (Gestão de Espaços de Tuplos e Pesquisa e Difusão de Dados).

Cada subsistema referido no parágrafo acima possui uma variável que mantém o valor do tempo total de execução de cada um. Não se confunda este valor com o tempo de simulação, pois um dado nó pode iniciar a sua execução apenas algum tempo depois de todos os outros. No suporte de pesquisa e difusão de dados, o tempo é essencial para definir a duração da tarefa e a taxa de amostragem, valores incluídos nos interesses. No caso do sistema de gestão de espaços de tuplos, a importância do tempo emerge na necessidade de avaliar a frescura de tuplos para, consequentemente, detectar a sua duplicação ou invalidez.

4.4.4 Suporte de Pesquisa e Difusão de Dados

Este suporte de pesquisa e difusão de dados para RSSF foi implementado conforme as linhas gerais do Directed Diffusion, descritas em [26], oferecendo duas operações: `startDiffusion`, `reinforceDiffusion`; e um evento: `receiveData`, como é visível na tabela 4.5. Na secção 4.4.4.1, apresentam-se os tipos e estruturas de dados que suportam o protocolo, descrevendo-se, no seguinte 4.4.4.2, o modelo de utilização do sistema, abordando a relacão técnica entre as 4 fases do protocolo e os seus elementos básicos de informação.

```
command DIFFIDT startDiffusion(tuple p, VLSIDT vlsId, BIGTIMET dur, ...
                                                    ... TIMET interval);
command void reinforceDiffusion(DIFFIDT diffId, BIGTIMET duration, ...
                                                    ... TIMET interval);

event result_t receiveData(Data *finalData);
```

Tabela 4.5: API desenhada para utilização do suporte de pesquisa e difusão de dados.

4.4.4.1 Elementos de Dados

Os elementos constituíntes da informação incluída e transmitida no suporte de pesquisa e difusão dividem-se em interesses e dados, representando, respectivamente, a informação pretendida e a informação concreta. De uma forma geral, há uma relação directa entre os tipos de dados deste protocolo com o sistema de gestão de espaços de tuplos, ao nível dos esquemas de "attribute-based naming". Nomeadamente, existe uma forte semelhança entre interesses e *patterns*, e entre tuplos e dados. A primeira relação justifica-se pelo facto de ambos corresponderem a informação incompleta, utilizada para suportar pesquisas. A segunda é evidenciada pelo facto de ambos possui-

rem apenas campos formais, isto é, informação exacta e concreta. A estrutura destes elementos de dados encontra-se, respectivamente, nas seguintes tabelas 4.6 e 4.7.

`typedef struct InterestStruct{`	
`TYPET type;`	indica que é um interesse
`NODEIDT source;`	identificador do nó origem
`NODEIDT vlsId;`	identificador da VLS
`NODEIDT vlsCommandId;`	comando executado na VLS
`NODEIDT lastNode;`	último hop do interesse
`TIMET interval;`	intervalo de frequência da amostragem
`BIGTIMET duration;`	duração da tarefa de amostragem
`SEQNUMT seqNum;`	número de sequência do interesse
`tuple t;`	tuplo pattern de pesquisa
`} Interest;`	

Tabela 4.6: Tabela representativa da constituição de um Interesse.

`typedef struct DataStruct{`	
`TYPET type;`	indica que são dados
`NODEIDT source;`	identificador do nó origem
`NODEIDT vlsId;`	identificador da VLS
`NODEIDT vlsCommandId;`	comando executado na VLS
`NODEIDT lastNode;`	último hop dos dados
`SEQNUMT seqNum;`	número de sequência relacionado com o do interesse
`SEQNUMT seqNumRun;`	retransmissão do número de sequência
`tuple t;`	tuplo de dados
`} Data;`	

Tabela 4.7: Tabela representativa da constituição singular dos Dados.

Os interesses e dados são mantidos no sistema de forma organizada, mediante a gestão de dois tipos de caches direccionados para cada tipo desses elementos, nomeadamente, da cache de interesses e da cache de dados. Na seguinte secção 4.4.4.2, descreve-se como estas estruturas suportam os processos de pesquisa e retorno de eventos, permitindo a detecção de duplicados e a formação da árvore de gradientes em direcção ao sink [26,27]. Adicionalmente, estas estruturas permitem o caching prolongado, agregação ou transformação de queries e dados para posterior envio, com vista a uma utilização mais eficiente dos recursos do sistema, mas as técnicas de agregação não são contempladas na presente implementação.

Repare-se que ambos os elementos possuem um campo `lastNode`, que contém o identificador do emissor da mensagem, que é necessário nalgumas fases do protocolo. A única forma aparente de um receptor determinar qual o emissor de uma mensagem recebida, na plataforma Mica2, é colocando esse valor no corpo (*payload*) da mensagem,

visto não existirem mais campos na estrutura de dados do TinyOS 1.x `TOS_Msg` para tal efeito [34, 35]. Os interesses ou dados são transmitidos directamente como corpo das mensagens do TinyOS, colocando-se um deles por meio de um apontador para o campo `data` da estrutura `TOS_Msg`, realizando o *cast* apropriado (`(Interest*)` ou `(Data*)`).

A seguinte imagem 4.5 ilustra a constituição de uma entrada de cada tipo de *cache* no sistema de pesquisa e difusão, nomeadamente, da *cache* de interesses e da *cache* de dados. No caso da *cache* de interesses, uma sua entrada possui o interesse, a variável de contagem decrescente para reenvio do interesse e o conjunto de gradientes formados, indicando quem lhe enviou o interesse. Relativamente a esta lista de gradientes, repare-se que o campo `duration` não foi criado, visto a duração do gradiente corresponder à do interesse, elemento que já possui o valor. A cache de dados encontra-se representada na figura 4.5, sendo cada entrada constituída pelo elemento de dados, pelo contador do próximo envio de dados e pelo conjunto de vizinhos de quem o nó em questão recebeu os dados. Esta estrutura mantém o estado dos eventos recentemente observados, permitindo assim a prevenção de ciclos de transmissão de dados.

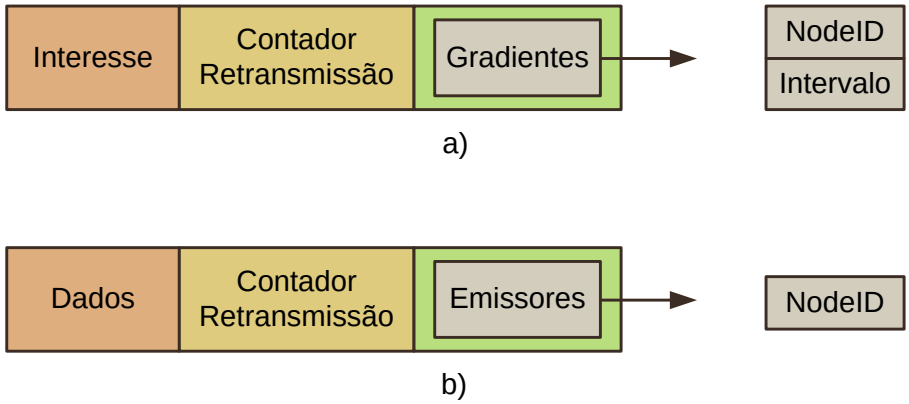


Figura 4.5: Constituição de cada entrada da *Cache* a) de Interesses e b) da *Cache* de Dados, com respectiva constituição de cada entrada do conjunto de gradientes e do conjunto de emissores de dados.

4.4.4.2 Método de Utilização e Operação do Sistema

Os métodos apresentados na tabela 4.5 do presente capítulo foram implementados de forma adequada ao ambiente de desenvolvimento e simulação apresentado em 4.1. Com base nessa API, a utilização do sistema é simples, sendo que é apenas necessário

executar o comando de início de uma pesquisa por difusão, e tratar o evento de recepção de dados oriundos dos nós source correspondentes. Durante a actividade de uma difusão, é possível, pelo método reinforceDiffusion, alterar-se a frequência de obtenção dos eventos devolvidos pela RSSF. A implementação dos processos que suportam esses métodos é descrita de seguida.

Início de uma Pesquisa por Difusão: através da chamada `startDiffusion`, o sistema permite realizar uma pesquisa sobre a RSSF. Ao se realizar este método, é colocado um interesse na *cache* de interesses do nó *sink*, que é inicialmente difundido para os seus vizinhos pelo método de comunicação *broadcast*. Ao longo do tempo, o ciclo infinito do sistema de pesquisa e difusão concretiza o *aging* da *cache* de interesses, que passa por decrementar dois campos - `sendIn` da entrada na cache e `duration` no interesse -, que mantêm o estado do interesse relativamente ao reenvio e validade do mesmo.

No caso da activação do refrescamento de interesses automático, quando o campo de reenvio alcança o valor zero, o interesse é de novo difundido pela rede, suportanto assim o processo automático de adição de novos participantes à pesquisa, ou removendo antigos participantes inactivos ou que já não correspondam a esse agrupamento. Ao se atingir o fim da validade de um interesse, este é ignorado pelo processo de *aging* da *cache*, e é considerado expirado, pelo que futuros dados correspondentes recebidos são descartados.

Difusão Intermédia de Interesses: um nó intermediário recebe uma mensagem contendo um Interesse e actualiza a sua *cache* de interesses. Nessa actualização, descarta as mensagens que se verificarem duplicadas, devido a casos de nós vizinhos também difundirem interesses. Não obstante a duplicação do interesse, o nó receptor faz uso do identificador do emissor dessa mensagem, para posterior formação dos gradientes, associando-o ao interesse contido na *cache*. A entrada nos gradientes é formada com o identificador do emissor e o valor `interval` contido no interesse recebido. Caso a mensagem de difusão do interesse não seja duplicada, o nó receptor continua a difusão realizando o *broadcast* da mesma, incluindo o seu identificador no campo `lastNode`, como se pode visualizar na estrutura do interesse na tabela 4.6, para que os seguintes nós receptores saibam quem a enviou.

Correspondência e Devolução de Dados: um nó *source* recebe e actualiza a sua *cache* com o interesse recém-chegado, e se não se tratar de um duplicado, ele também continua a difusão do mesmo. Adicionalmente realiza a comparação *one-way match* do mesmo com todos os elementos da *cache* de dados, e caso encontre correspondência, nomeadamente entre os tuplos de cada um, envia o elemento de dados descoberto

para quem lhe enviou o interesse. Este envio de dados repete-se ao longo da duração da tarefa enquanto activa, indicada com o interesse recebido, pelo que a *cache* de dados vai sendo actualizada conforme o estado do espaço de tuplos do nó.

Devolução Intermédia de Dados: na recepção de uma mensagem de dados por parte de um nó intermediário, este realiza a sua correspondência com um das entradas da sua *cache* de interesses. Caso obtenha uma entrada que possua um interesse que faça correspondência com os dados recebidos, o intermediário envia para um, ou mais gradientes, a mensagem de dados recebida. Tal como na difusão de interesses, o nó intermediário altera o campo `lastNode` com o seu identificador.

Reforço de uma Pesquisa: na chegada da primeira mensagem de dados ao nó *sink*, este reforça uma, ou mais, das ligações que possui com os seus vizinhos. Estes vizinhos são descobertos com o envio de uma mensagem de dados anteriormente realizado por eles mesmos, em direcção ao *sink*, que por sua vez constrói os emissores de uma determinada entrada na *cache* de dados. O *sink* envia uma mensagem endereçada a esses nós, contendo um interesse com valores de `interval` e `duration` diferentes. O processo de reencaminhamento do reforço é concretizado usando uma metodologia de escolha do melhor vizinho emissor de dados, pelo que essa mensagem chegará, eventualmente, ao nó *source* que capturou os dados em questão, alterando, consequentemente, a sua taxa de amostragem e de envio de eventos.

4.4.5 Suporte de Espaços de Tuplos

O suporte à gestão de espaços de tuplos presente no MW das VLS é concretizado mediante 3 componentes TinyOS, à semelhança da arquitectura figurada nos componentes do TeenyLIME ilustrados na imagem 3.3. A API deste sistema mantém-se inalterada, face ao sistema TeenyLIME original existente em [5], exceptuando na alteração do parâmetro referente ao escopo da operação, como explicado na secção 4.4.1. No entanto, não foi possível esse sistema ser aplicado directamente na presente dissertação, pelo que a sua implementação foi baseada numa adaptação para a versão 1.x do TinyOS, visto a sua publicação estar apenas dirigida para a versão 2.x desse sistema operativo.

O sistema de espaços de tuplos é utilizado através do componente `TupleSpaceM`, que oferece uma API composta pelo segundo grupo de métodos apresentados na tabela 4.3. A sua função principal é direccionar o fluxo de execução para um dos subcomponentes abaixo de si: `LocalTeenyLIME` e `DistributedTeenyLIME`, tendo em conta o valor do parâmetro `vlsId`. Outra funcionalidade, mas transparente ao utilizador, é a atribuição automática de identificadores de operação, devolvido por parâmetro for-

necido em cada operação sobre espaços de tuplos, nomeadamente pelo `opId`.

4.4.5.1 Componente de Suporte ao Espaço de Tuplos Local

Este componente surge como suporte às operações sobre o espaço de tuplos local ao nó onde o MW está presente, bem como realiza a gestão do mesmo, gerindo a frescura dos tuplos, bem como da estrutura que os comporta. Os tuplos são mantidos num *array*, cuja entrada está associada a uma variável booleana `deleted`, que indica a invalidade do tuplo.

A gestão do *array* de tuplos é simples. Um tuplo é inserido na primeira entrada marcada como contendo o primeiro tuplo inválido ou o mais antigo. A remoção de tuplos, devido à chamada `in` ou `ing`, simplesmente torna a entrada do pretendido como inválida, colocando a variável `deleted` a verdadeiro.

A gestão da frescura dos tuplos existentes no espaço local é realizada ao longo do ciclo do sistema, regulado pelo temporizador descrito na secção 4.4.3. Num intervalo de tempo com a duração de apenas alguns ciclos, é despoletada uma pesquisa pelos tuplos que já possuem um nível de frescura admitido. Através de uma pesquisa linear ao *array*, caso cada tuplo válido possua uma estampilha temporal de criação demasiado antiga face a um valor de referência da frescura dos dados, a entrada é marcada como inválida.

Adicionalmente, é neste componente que é realizada a gestão de reacções ao surgimento de tuplos no espaço local. As reacções instaladas são organizadas num *array* cujas entradas contêm a *pattern* que define a correspondência com o tuplo pretendido, uma estrutura de operação TeenyLIME do tipo `TLOpId_t` e um booleano para indicar o disparo único da reacção. Em cada inserção de um tuplo no espaço local, se se verificar a correspondência entre o *pattern* de uma reacção instalada e esse novo tuplo, a reacção dispara, e o tuplo é devolvido ao sensor que instalou a reacção, de forma local ou remota.

Relativamente às operações despoletadas de forma remota, isto é, pelo *sink* de uma VLS à qual um dado nó pertenca, essas são recebidas pelo componente Distributed-TeenyLIME, que, mediante a interface BridgeTupleSpace oferecida pelo componente local, permite a sua aplicação no espaço de tuplos do nó receptor. No caso da colocação de um tuplo, este é visto como se tratasse de um tuplo colocado de forma local. No caso das reacções, estas são colocadas igualmente na mesma estrutura das locais, mas com a diferença que o seu campo `TLOpId_t` as assinala de origem externa - o campo `msgOrigin` da estrutura de operação na tabela 4.8 indica o identificador do nó que instalou a reacção, bem como o destino da mensagem de resposta que conterá o tuplo correspondente -.

`typedef struct {`	
`uint8_t commandId;`	identificador da operação
`NODEIDT msgOrigin;`	identificador do nó origem da operação
`} TLOpId_t;`	

Tabela 4.8: Representação da estrutura de operação `TLOpMsg_t`, original do sistema TeenyLIME.

4.4.5.2 Componente de Suporte ao Espaço de Tuplos Distribuído

O subsistema de espaços de tuplos distribuídos concretiza o suporte às operações sobre os espaços externos ao nó em questão, e o tratamento de consequentes respostas. A sua interface de programação é igual à do sistema TeenyLIME geral, oferecendo a possibilidade de manipulação de espaços remotos, bem como a instalação de reacções ao surgimento de tuplos nos mesmos. De forma concreta, o issuing de cada tipo de operações consiste directamente no envio de uma mensagem indicando a operação e o tuplo, ou *pattern*, correspondente.

Desta forma, parte do processamento deste componente passa pela análise das mensagens recebidas, e correspondente chamada do método na interface BridgeTupleSpace. A aplicação desse método no espaço local pode resultar numa resposta, comunicada através do *signal* de evento na interface de interligação, para posterior envio ao dispositivo originador da operação remota.

4.4.6 Enlace dos Suportes de Espaços de Tuplos e de Pesquisa e Difusão

Tendo-se os dois componentes principais implementados (Gestão de Espaços de Tuplos e Sistema de Pesquisa e Difusão de Dados), coloca-se um terceiro, que actua como ponto de utilização do sistema de comunicação, não só por parte da aplicação, mas também do componente de espaços de tuplos remotos. Este componente, denominado Gestor de VLS, possui, adicionalmente, a funcionalidade de gestão dos agrupamentos lógicos criados pelo próprio nó e a manutenção do estado aos quais pertence.

Havendo elementos similares dos dois componentes principais, algumas estruturas de dados encontram-se agora proximamente relacionadas. Tal é o caso da *cache* de dados e o *array* do espaço local, que possuem em comum os tuplos. Desta forma, a *cache* terá, ao invés da informação concreta, um apontador para o tuplo correspondente, existente no espaço de memória do *array* de tuplos. Dado este esquema simples de partilha de memória, quando o sistema de espaço de tuplos - por acção de um `in` ou `ing` local ou remoto -, remove um tuplo, a entrada correspondente no espaço é

marcada como eliminada, e o apontador tornado inválido, com posterior remoção da *cache* de dados.

A ligação entre os três componentes inicia-se com a ligação entre o Gestor de VLS e o Sistema de Pesquisa e Difusão de Dados. A operação `startVLS` relaciona-se directamente com o método `startDiffusion` do Suporte de Comunicação do MW, cujo controlo é dado directamente à aplicação por meio da interface `VLSManager`, que contém os métodos de gestão de VLS da API das VLS. Ao suporte a espaços de tuplos, é disponibilizada a interface `VLSTupleSpace`, que permite que sejam despoletadas as operações de espaços de tuplos sobre uma dada VLS, e devolvidas as consequentes respostas, por meio do *signal* de um evento.

Como referido anteriormente na secção 4.4.4, a informação contida no sistema de pesquisa e difusão de dados surge na forma de tuplos, elementos de campos tipificados e oriundos do modelo de espaços de tuplos empregue na arquitectura concretizada neste trabalho. No entanto, tal não vem especificado nas linhas descritivas do Directed Diffusion, onde é usado um esquema de nomeação de dados do tipo chave-operação-valor, que se substituiu por razões de economia dos recursos de memória e impacto nas comunicações, mas, principalmente, pela razão das aplicações serem dirigidas ao paradigma de espaços de tuplos, resultando em menos alterações no código por substituição do tipo de informação manipulada.

4.5 Modelo de Energia

A obtenção das medições de consumo energético é realizada com base num *plugin* do TinyViz, denominado Power Profile, apresentado anteriormente em 4.1. Este plugin foi produzido pelos autores do PowerTOSSIM, uma ferramenta de pós-processamento do *output* de aplicações executadas no simulador, realizando os cálculos necessários para obter os consumos de cada nó. No caso do *plugin* Power Profile, é seguida a aproximação de captura de *prints*, feitos pela aplicação, e posterior análise dos mesmos. De uma forma geral, todos os componentes principais de um dispositivo, como o CPU, Rádio, Leds e EEPROM (Memória), fazem o *print* de cada mudança de estado e operação, e o *plugin* captura e acompanha essas mudanças. Por exemplo, quando o componente Rádio é usado para iniciar uma transmissao, faz o *print* de informação do início dessa tarefa, e quando esta terminar (para se dar inicio a outra), faz outro *print*. Assim, é possível medir-se a duração de cada operação, para se calcular o custo da sua execução, através da seguinte fórmula:

$$\text{Poder Energético} = W = I * V \ ,$$

$$\text{Consumo Energético} = J = W * t \ ,$$

sendo I a intensidade de corrente eléctrica que a operação exige definida no modelo de consumo energético, V a voltagem da plataforma de sensores alvo, e t a duração de execução da tarefa.

A intensidade de corrente eléctrica de cada operação é definida num modelo de energia igualmente referente à plataforma empregue, sendo neste caso os sensores Mica2, visto não ter sido possível obter-se um modelo de energia dos dispositovos Mica tão detalhado como o presente no Power Tossim [52].

5

Validação Experimental

O actual capítulo apresenta o conjunto de testes realizados sobre a arquitectura implementada na presente dissertação, para validação das várias contribuições introduzidas pelo sistema de VLS, face ao impacto da sua aplicação numa RSSF. Sendo o sistema concretizado em vários níveis, foi necessário realizar a validação de cada um deles, garantindo às camadas superiores os vários pressupostos objectivados (segurança, coerência, fiabilidade) por cada nível inferior. Tal como os anteriores capítulos do presente documento, a análise realizada nesta secção de validação segue uma aproximação bottom-up.

Na primeira secção 5.1, descrevem-se as quais as contribuições obtidas e os custos impactuais de implantação do sistema de VLS, por meio de métricas de validação qualitativas e quantitativas. Na secção 5.2 indica-se os vários parâmetros de configuração do ambiente de desenvolvimento e simulação, indicando como está estabelecido e como realizar a execução de aplicações. A terceira secção 5.3 aborda um conjunto de validações preliminares à validação do MW das VLS, analisando os níveis abaixo do mesmo, constituintes da infra-estrutura base empregue.

Depois, a secção 5.4 inclui a validação experimental da arquitectura das VLS, realizando uma medição dos custos impactuais da sua aplicação em RSSF, e analisando os níveis de segurança e resiliência alcançados com a utilização do sistema proposto. A obtenção destes indicadores de impacto foi dividida em dois tipos de testes: comparação da implementação das VLS com a solução base do sistema de gestão de tuplos suportado por um mecanismo simples de comunicação por difusão, e obtenção dos parâmetros que conferem uma performance aceitável para dimensões da rede elevadas. Este capítulo termina com a secção 5.5, que inclui uma síntese crítica dos resultados obtidos na avaliação do sistema das VLS face às contribuições da dissertação.

5.1 Contribuições do Sistema de Vizinhanças Lógicas Seguras

Como foi introduzido no capítulo 1.5, o conjunto de contribuições do presente trabalho engloba a facilidade de produção de aplicações, a segurança e a fiabilidade dos dados circulantes na RSSF, com a utilização de um suporte de *middleware* instanciando o paradigma das VLS. A introdução deste MW nos dispositivos de uma RSSF vem, obviamente, com um custo adicional. Tal hipotese é justificada pelo facto de se realizarem, neste MW, operações de processamento e de armazenamento adicionais, face às incluídas nos sistemas base, e com os quais a arquitectura das VLS foi comparada.

Numa visão qualitativa, pretende-se com a arquitectura implementada fornecer os três pontos contributivos - apresentados no parágrafo anterior - de igual, ou melhor, forma que a solução do TeenyLIME. Relativamente à produção de programas, a expressividade mantém-se a mesma, visto a arquitectura das VLS incluir a mesma interface de utilização de espaços de tuplos que o sistema original, sendo que apenas acrescenta as funcionalidades de gestão das VLS. A tabela 5.1 ilustra esta comparação dos sistemas sobre os três indicadores qualititativos.

Indicadores Qualitativos	TeenyLIME	VLS
Expressividade de Programação	Paradigma das VL	
Segurança das Comunicações	Não	Sim (TinySec e DD)
Fiabilidade das Comunicações	Não	Sim (TinySec e DD)

Tabela 5.1: Comparação do sistema TeenyLIME com a arquitectura das VLS com base nos indicadores de avaliação qualitativa.

Ao nível das contribuições de fiabilidade do sistema e segurança dos dados transmitidos ao longo da rede, a arquitectura das VLS possui vantagem face ao sistema base do TeenyLIME. Tal é justificado por o sistema de VLS operar sobre uma camada de comunicação segura (TinySec) associada à pilha de comunicação standard do TinyOS, não havendo diferenças no porting de qualquer uma. Adicionalmente, o sistema das VLS possui mecanismos de resiliência pro-activa em si incorporados, devido ao seu sistema de pesquisa e difusão de dados. O TeenyLIME não alcança tais propriedades, pois opera directamente sobre a pilha de comunicação standard, que por si só não contemplam o suporte de pressupostos de segurança.

Relativamente à análise do impacto de utilização deste sistema de VLS, surgem como métricas interessantes dessa avaliação: a latência, a fiabilidade e coerência dos espaços de tuplos, e o consumo energético. A necessidade de observação das alterações de latência é considerada, numa primeira abordagem, a métrica que produzirá maiores

diferenças, visto a camada de MW das VLS incorrer em processamento adicional. Com a medição da fiabilidade das comunicações, pretende-se verificar se o sistema produz um comportamento correcto e coerente, não se tornando a si próprio inválido por falta de completude das operações. Finalmente, o estudo sobre o consumo energético permite realizar a observação do quanto se ganha em se realizar processamento adicional - introduzido, principalmente, pelo componente de pesquisa e difusão de dados da pilha das VLS - para se evitar mensagens e outra informação desnecessária. Toda esta verificação será descrita na secção 5.4.2, mas primeiro, introduz-se a secção 5.2 relativa ao ambiente de desenvolvimento e teste e a secção 5.2.3 que compreende os vários parâmetros estabelecidos para os diferentes cenários de validação experimental.

5.2 Configuração do Ambiente de Desenvolvimento e de Simulação

O ambiente de desenvolvimento e simulação experimental dos sistemas concebidos tem como base a configuração descrita na secção 4.1 do capítulo anterior. Recapitulando a constituição do ambiente de desenvolvimento e simulação, sintetizado na tabela 5.2, foi desenvolvida uma solução correspondendo a uma organização de serviços ao estilo do sistema descrito no capítulo 3, codificada em linguagem NesC, testada experimentalmente no ambiente do sistema operativo TinyOS 1.x, oferecido pela distribuição XubuntOS. Adicionalmente, foi utilizada como ferramenta de visualização dos eventos de simulação o módulo TinyViz, uma implementação em Java que acompanha a versão do sistema operativo utilizado. A extensibilidade do TinyViz vem na forma de plug-ins, desenvolvidos igualmente em Java, e permitindo, por exemplo, realizar capturas de eventos específicos conforme o programador assim o desejar. Na seguinte subsecção, apresentam-se os plugins desenvolvidos especificamente para o trabalho da presente dissertação, cuja relação com os aspectos de validação experimental é directa.

Funcionalidades	Sistema
Visualizador Gráfico de Eventos	TinyViz
Emulador / Simulador de Eventos	TOSSIM
Suporte Java	1.5.0
Liguagem de Programação	NesC 1.1.3
Sistema Operativo RSSF	TinyOS 1.0.1

Tabela 5.2: Caracterização geral do ambiente de desenvolvimento e simulação utilizado na presente dissertação.

5.2.1 Auxiliares de Captura de Eventos na Visualização de Simulações

Tendo em mente a extensibilidade oferecida pelo módulo de visualização gráfica TinyViz, foram produzidos neste trabalho vários *plugins* com os seguintes objectivos: auxiliar na análise do desenvolvimento da plataforma de MW das VLS, actuando como um *debugger* ao longo da execução de aplicações; suportar a obtenção e medição específicas dos vários indicadores que compõem a avaliação quantitativa; verificar a completude de operações específicas aos sistemas que compõem a arquitectura.

O papel dos *plugins* como auxiliar de captura de eventos, activo durante a execução de uma aplicação, baseia-se no princípio intrínseco ao TinyViz, como introduzido na secção 4.1: a aplicação executada pelo TOSSIM realiza prints com palavras-chave específicas, aos quais a restante informação associada é concatenada, e cada *plugin* captura as *keywords* para quais foi desenvolvido, para obter informação sobre os eventos em causa. Assim, é possível determinar-se a ocorrência de um evento específico, quando é que esse mesmo ocorreu, e qual a informação a que lhe corresponde. Adicionalmente, é ainda possível capturarem-se eventos de simulação de um ou mais nós em específico, condicionando-se a visualização da simulação dos mesmos. Esta escolha pode ser alterada durante a execução de uma aplicação, e permite, por exemplo, o *debug* de uma dada área da rede, descartando eventos dos restantes participantes da rede.

Desta forma, desenvolveram-se três *plugins* específicos, destinados às principais métricas quantitativas de validação do MW das VLS, da seguinte forma:

- Para se concretizar a medição dos indicadores de **latência**, o sistema implementado na presente dissertação realiza um *print* no início da transmissão de toda e qualquer mensagem, pelo que o nó destinatário realiza outro *print* no instante em que inicia a recepção da mesma. O *plugin* de medição da latência memoriza qual o tempo de simulação do *print* de início da transmissão, pelo que quando ocorrer o *print* de início da recepção dessa mesma mensagem, é subtraído o tempo actual ao tempo de ocorrência desta última. De modo a se saberem quais os intervenientes do processo de comunicação, a informação sobre o emissor e receptor (identificadores dos nós) é também colocada nos prints, podendo também associar qual o número de sequência de cada mensagem relacionada, ou até outro tipo de informação relacionada.

$$\text{Latência} = T_{\text{início recepção}} - T_{\text{início envio}}$$

- Para a obtenção dos níveis de **fiabilidade**, é realizada a captura do envio e da recepção de cada mensagem, por meio de dois *prints* distintos. É mantido, para

cada nó, o número de mensagens enviadas, monitorando, adicionalmente, quais e quantas mensagens alcançam o seu destino. Após a execução da aplicação, possui-se o número de mensagens enviadas por cada elemento emissor e o número de mensagens recebidas por cada receptor dessas mensagens, calculando-se assim o racio de fiabilidade. Cada nó emissor e receptor realiza o *print* do evento em questão com uma *keyword* alusiva, respectivamente, ao envio e à recepção de uma mensagem, associando a restante informação necessária.

$$\text{Fiabilidade}_{\text{nó A}} = \text{N}_{\text{msgs enviadas por A}} \ / \ \text{N}_{\text{msgs recebidas oriundas de A}}$$

- Relativamente à medição do **consumo energético**, são capturados no *plugin* do TinyViz os eventos com uma *keyword* relativa à energia, os quais são produzidos pela simulação ao se empregar a flag $-p$ do TOSSIM. Recapitulando o método descrito na secção 4.5, cada evento indica qual foi a operação que comecou a ser realizada, bem como qual o nó em questão, sendo que o *plugin* mede a duração da mesma para finalmente calcular o consumo energético. Adicionalmente, o *plugin* cria e preenche uma grelha de valores energéticos, possível de ser copiada para o *clipboard* e colocada numa ferramenta de manipulação de folhas de cálculo, para melhor análise, organização e produção de gráficos relativos à informação de gastos energéticos obtidos numa simulação.

Para além dos auxiliares acima descritos, cujo propósito está relacionado com a medição de indicadores relativos à validação experimental, foram desenvolvidos outros plugins que permitem visualizar o estado das estruturas de dados mais importantes, tais como as caches do sistema de pesquisa e difusão de dados ou os espaços de tuplos de cada nó em específico.

5.2.2 Topologias e Configurações de Redes Empregues

A tabela 5.3 ilustra os quatro cenários utilizados para simulação experimental da pilha de VLS, onde todos estes partilham as mesmas características de distribuição, exceptuando a dimensão da rede e, consequentemente, o número de nós Source, que corresponde à metade da dimensão. Na figura 5.1 apresentam-se as quatro principais topologias, referentes aos cenários da validação experimental da presente dissertação: disposição em grelha, com 3x3, 5x5, 7x7 ou 9x9 nós, e com sink no centro, tendo-se um número de participantes da VLS constante, correspondendo sempre à metade da rede ($(N-1)/2$). Estas topologias fazem parte dos primeiros conjuntos de testes, pelo que foram acrescentadas, mais tarde, outras escalas de rede intermédias, nomeadamente contendo 15, 20, 35, 55, 65, 90, 100 e 125 nós.

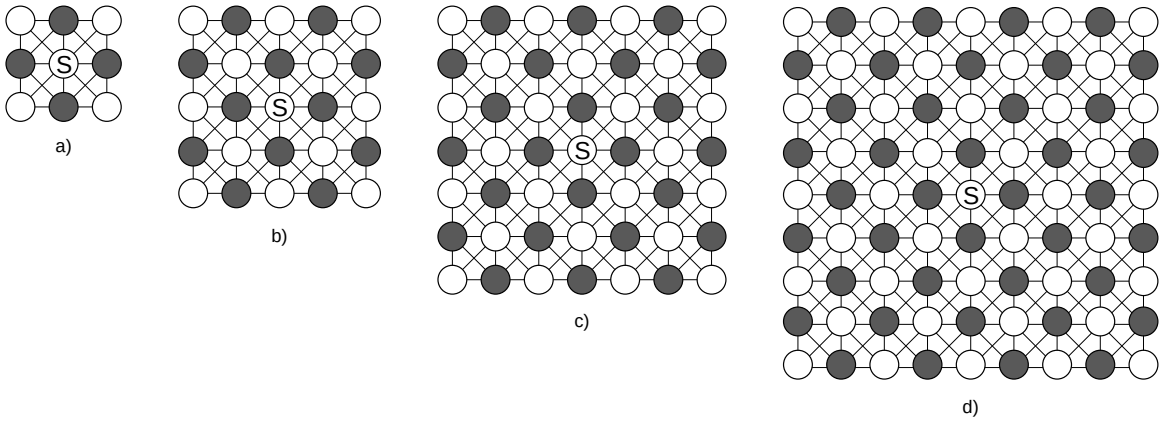


Figura 5.1: Figura representativa das quatro principais topologias empregues na validação experimental: a) 3x3, b) 5x5, c) 7x7 e d) 9x9. O nó Sink encontra-se no centro assinalado com um S, e os nós Source - participantes da VLS - estão a cinzento. Cada aresta representa a presença de conexão entre os dois nós ligados.

Cenário 9 nós	Grelha com 3x3 nós, 1 nó Sink no centro, 4 nós Source uniformemente espalhados
Cenário 25 nós	Grelha com 5x5 nós, 1 nó Sink no centro, 12 nós Source uniformemente espalhados
Cenário 49 nós	Grelha com 7x7 nós, 1 nó Sink no centro, 24 nós Source uniformemente espalhados
Cenário 81 nós	Grelha com 9x9 nós, 1 nó Sink no centro, 40 nós Source uniformemente espalhados

Tabela 5.3: Descrição dos quatro cenários definidos para a realização do primeiro conjunto de testes que compõem a validação experimental.

5.2.3 Parâmetros dos Testes Experimentais

A seguinte tabela 5.4 indica quais os parâmetros de teste, variáveis ou constantes, para validação experimental do MW de VLS, e os seus valores possíveis. Estes parâmetros são aplicados em todos os conjuntos de testes (verificação experimental comparativa e obtenção da dimensão da rede face ao número de retransmissões). A tabela 5.5 apresenta os parâmetros adicionais a ter em conta no segundo conjunto de testes - verificação da melhor dimensão da rede num cenário com retransmissões -.

Tamanho Rede	9, 25, 49 ou 81
Número de Sinks	1
Número de Sources	4, 12, 24 ou 40
Número de Ensaios	10
Máximo de Espera por Eventos	65s, 125s
Densidade dos nós	Homogénea
Área de Cobertura	100 x 100 metros
Número de Vizinhos	mínimo 3, máximo 8
Modelo de Rádio	Raio Fixo Disc10, factor de escala 0.7
Tamanho Buffer Envio	50
Slots BufferedSend	10
Duração Espera BufferedSend	min: 0.5s, méd: 2.5s, máx: 5s
Limite de Throughput	800bps, 160bps, 80bps
Mobilidade	Estacionária
Tipo de RSSF	Homogénea
Protocolo MAC	IEEE 802.15.4
Modelo de Ruído	Definido pelo TOSSIM
Throughput do Emulador	TOSSIM 10kbps default

Tabela 5.4: Parâmetros constantes ou variáveis que suportam as execuções dos testes de validação experimental.

Dos parâmetros descritos na tabela 5.4, é importante salientar o número de ensaios, que diz respeito aos concretizados com sucesso. Como se verá na secção 5.4.2.1, a fiabilidade dos envios de eventos originados pelos nós source pode divergir da total completude da mesma, isto é, dos 100%, pelo que é necessário definir-se um número mínimo e aceitável de ensaios com sucesso, ou seja, onde se conseguiu obter eventos de todos os participantes de uma VLS. Desta forma, é essencial definir-se um tecto máximo de tempo de espera dos eventos enviados, por forma a limitar-se a infinitude das simulações e obterem-se valores de consumo de energia razoáveis. Este valor máximo de espera de eventos é definido conforme o sistema/cenário em utilização, pois espera-se que um será mais rápido a obter o maior número possível de participantes que o outro. Reserva-se enfoque na escolha do tempo de espera aleatório para envio das mensagens de saída em buffer, que variam entre 0,5 e 5 segundos, dividido em

10 slots, que, por uma série de observações preliminares, verificou-se ser o mais adequado: tempos de espera menores conduziam a simulações mais rápidas mas menos fiáveis, isto no caso de redes com dimensão significativamente elevada ($\geq$ 50 nós).

Tamanho Rede	81, 121, 169 ou 225
Número de Sources	40, 60, 84 ou 112
Número de Ensaios	5
Máximo de Espera por Eventos	65s, 125s, 145s, 200s
Duração Espera BufferedSend	0s
Limite de Throughput	800bps

Tabela 5.5: Parâmetros adicionais definidos para o segundo conjunto de testes relativo à obtenção de uma dimensão da rede face ao número de retransmissões efectuadas por evento capturado.

Da tabela 5.5, para o segundo tipo de testes, é relevante salientar que o tecto máximo de espera por eventos foi obtido por observações preliminares aos ensaios em concreto, pelo que é um valor com uma tolerância na ordem dos 25 segundos após a recepção do último participante a juntar-se à vizinhança. O número de ensaios reduziu-se, comparativamente com o parâmetro correspondente do primeiro conjunto de testes, devido ao elevado tempo real necessário para concretizar cada ensaio dados valores elevados de dimensão da rede. Relativamente ao tempo de espera entre envios de mensagens de saída em buffer, este parâmetro foi reduzido para 0 segundos mas, tendo em mente que o sistema de MW das VLS possui um ciclo de simulação que progride em intervalos de 500 ms, como visto na anterior secção 4.4.3, o limite teórico de throughput alcança os 800bps, para envios sucessivos de mensagens.

Acrescentou-se também, a este grupo de testes de retransmissão de eventos, um maior número de escalas de rede testadas, não só as dimensões acrescentadas ao primeiro conjunto de testes, 15, 20, 35, 55, 65, 90, 100 e 125 nós, mas também valores acima dos 125, nomeadamente, 150, 200, 225 e 256 nós.

5.3 Validações Preliminares

Pretende-se, nesta secção, dar início à análise *bottom-up* proposta na introdução deste capítulo, que passa por apresentar uma validação dos sistemas que constituem a infra-estrutura base, TinySec sobre TinyOS, na qual o protótipo desenvolvido na presente dissertação assenta.

Têm-se em conta os resultados experimentais obtidos com a utilização do módulo TinySec, quer por parte dos autores em [32], como por parte de outros investigadores [40], pelo que se assume, para já, que o TinySec serve adequadamente como sistema

de comunicação segura para a infra-estrutura base do MW de VLS, incrementando um impacto mínimo na performance e recursos computacionais de uma RSSF onde seja aplicado.

Os sistemas de Directed Diffusion e TeenyLIME não foram testados previamente, visto não existir, na altura corrente da dissertação, uma implementação em nesC para TinyOS versão 1.x, pelo que: o sistema de pesquisa e difusão de dados foi implementada de raiz com base nas linhas gerais do artigo principal do Directed Diffusion [27] e dissertação de doutoramento do mesmo autor [26]. O sistema TeenyLIME foi novamente lançado pelos seus autores sensivelmente a meio da presente dissertação, mas direccionado para a versão 2.x do sistema operativo de RSSF, pelo que foi necessário realizar uma adaptação, quase que completa, do sistema original para a versão mais antiga do TinyOS, a 1.0.1.

5.4 Validação da Arquitectura

Nesta secção, são validadas as contribuições da aproximação concretizada nesta dissertação, realizando-se uma avaliação dos custos impactuais do uso da arquitectura das VLS, e uma análise comparativa da mesma com outros sistemas ou aproximações. São usadas como métricas de impacto a latência das comunicações, a fiabilidade do envio de mensagens, e os gastos energéticos emergidos com a utilização do sistema implementado.

5.4.1 Sistemas Empregues na Validação Experimental

Pretende-se efectuar uma validação em paralelo do sistema de VLS com uma aproximação que mantenha muitas das características do sistema original TeenyLIME, acrescentando a importância central de possibilidade de pesquisa dos participantes de uma VL. Outro objectivo em mente é não criar um sistema demasiado sofisticado, por forma a não se usarem mecanismos similares aos do suporte de difusão da pilha de VLS, tais como os mecanismos de cache e de resiliência multi-rota.

Foi assim desenvolvido um sistema composto por uma pilha estratificada, ao estilo da desenvolvida para as VLS, mantendo-se o suporte de gestão de espaços de tuplos, e, consequentemente, a API oferecida com esse propósito. A diferença deste sistema passa pela camada de serviços de pesquisa e difusão de dados, onde se emprega, como suporte de comunicação e de gestão de VLS, um sistema muito mais simples denominado `SimpleBroadcast`.

Tendo em mente a característica de não se adicionar demasiada complexidade ao sistema original do TeenyLIME, o modelo de utilização passa, neste sistema de Simple-

Broadcast, por associar-se o tuplo, correspondente à definição de uma VL, a cada operação de manipulação de um espaço de tuplos. Mais concretamente, a indicação de que o escopo da operação abarca o nível de uma vizinhança (argumento `TLNEIGHBORHOOD`) indica, adicionalmente, que é transportado, com os restantes argumentos da primitiva, o tuplo que define a VL.

O seu método de operação simplificado emprega somente comunicação por *broadcast* (uma das formas de comunicação em TinyOS) e um mecanismo básico de filtragem de mensagens duplicadas por meio do acompanhamento por número de sequência. Este sistema possui duas fases de utilização: difusão de uma operação sobre o espaço de tuplos dos participantes de uma VL e difusão de resposta de uma operação recebida, caso o processamento da mesma no nó alvo possua retorno.

A combinação do sistema de *broadcast* simples é concretizada directamente com as operações externas do componente de espaços de tuplos distribuídos. Cada envio de uma primitiva a aplicar sobre uma vizinhança é realizada com a disseminação de uma mensagem que, para além de incluir a mensagem original do sistema base TeenyLIME, inclui informação sobre quem a originou - ou seja, o criador da VL -, qual o seu destino - que pode ser todo e qualquer nó ou um nó em particular -, e qual o número de sequência da mensagem.

Na recepção de uma mensagem, cada nó compara o número de sequência da mesma com o da última mensagem válida anteriormente recebida, descartando assim as duplicadas. De seguida, o receptor verfica se o destino da mensagem é o seu endereço, e caso seja, informa o sistema de gestão de espaços de tuplos distribuído da chegada de uma mensagem de tuplo. Esta característica vem ao encontro da aproximação original do TeenyLIME, onde o componente de processamento distribuído de espaços de tuplos é responsável por efectuar o handling das mensagens recebidas. Finalmente, se a mensagem não for duplicada, o receptor torna-se igualmente um intermediário no processo de difusão da mesma, realizando a sua retransmissão para os seus vizinhos.

5.4.2 Avaliação do Impacto do uso das Vizinhanças Lógicas Seguras

Como foi anteriormente referido na secção 5.1 inicial deste capítulo, era esperado o surgimento de custos adicionais ao se empregar o MW das VLS. A modelação da arquitectura justifica a introdução desses custos, tendo em conta que são adicionados processamentos e dados adicionais ao sistema de gestão de espaços de tuplos. São assim avaliados, num primeiro conjunto de testes, os incrementos ao nível das comunicações, como latência na criação e obtenção de eventos de uma VLS; ao nível da gestão de espaços de tuplos, como a fiabilidade dos dados transmitidos ao longo da rede; ao nível do consumo energético, por ser um dos recursos mais importantes nas RSSF, e se

esperarem melhorias dadas as vantagens dos sistemas utilizados pela pilha das VLS. A fiabilidade dos dados transmitidos é também observada para diferentes valores de retransmissão dos mesmos. Finalmente, é realizada uma avaliação, de forma singular ao sistema das VLS, dos níveis de segurança obtidos e do comportamento do novo módulo de comunicação usado pelo sistema de gestão de tuplos.

5.4.2.1 Avaliação de Indicadores de Latência

Nesta série de ensaios de observação dos indicadores de latência, foi realizada uma comparação da execução da aplicação de teste experimental, utilizando-se o sistema de Difusão Simples face ao sistema das VLS. Segundo a definição geral, a latência de criação e obtenção de eventos de uma VLS corresponde ao tempo decorrido entre o início do pedido de criação da vizinhança, ao nível da aplicação, até ao tempo de chegada da resposta do último participante da VLS, ou seja, até o sink receber, pelo menos, uma resposta de todos os nós que pertencem, concretamente, à vizinhança.

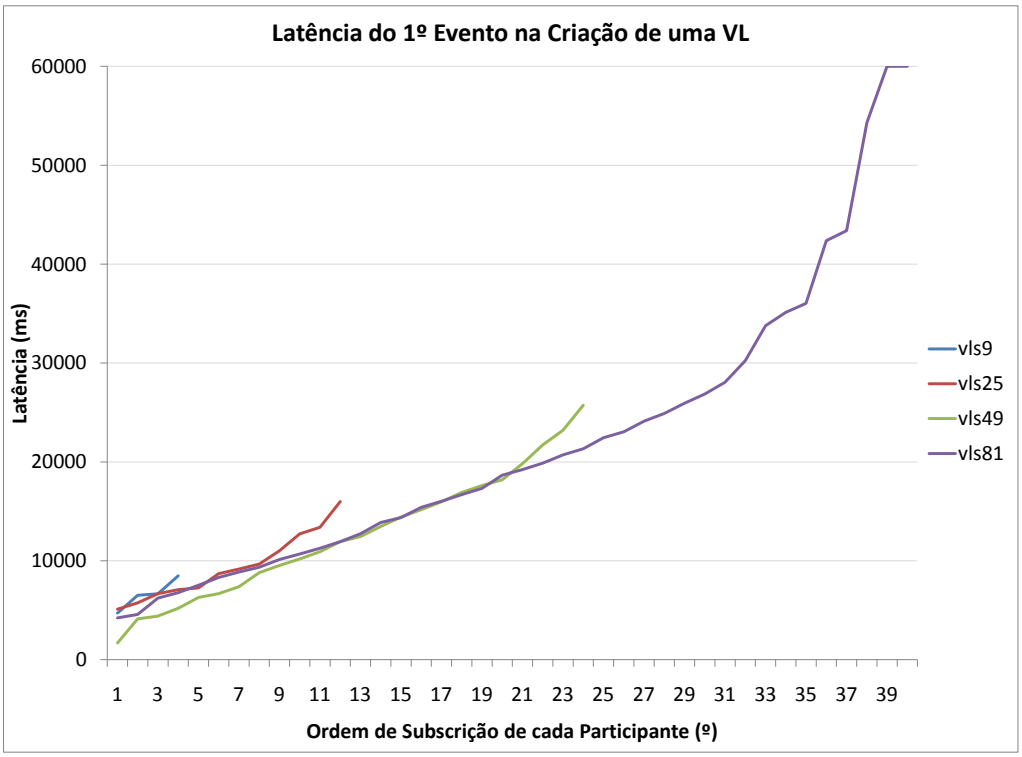


Figura 5.2: Valores de latência obtidos, com a aproximação das VLS, para cada participante de uma VL descoberto em ordem crescente de tempo.

A figura 5.2 representa o gráfico dos 10 ensaios realizados utilizando-se o sistema de VLS como base da aplicação, face a várias dimensões de rede. Pode-se verificar que

a taxa de evolução é similar para todas as dimensões, exceptuando a recepção de cerca de 8% dos últimos participantes a responderem. Tal significa a tendência da taxa de crescimento da latência manter-se-á para valores mais elevados de escala.

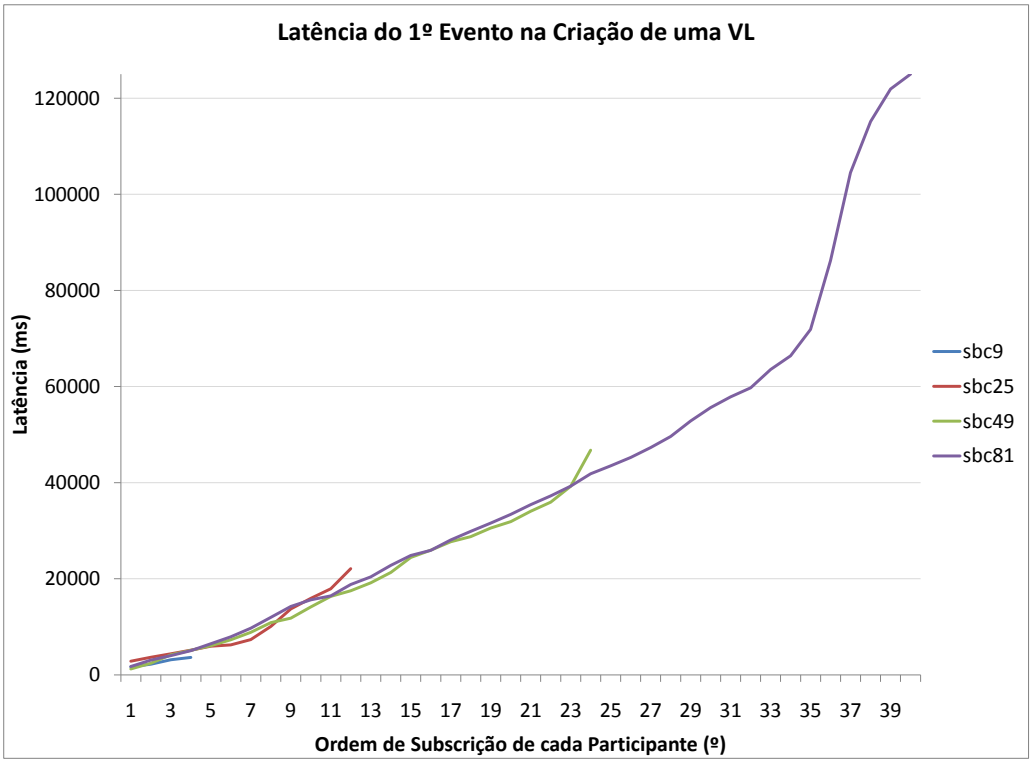


Figura 5.3: Valores de latência obtidos, com a aproximação de Difusão Simples, para cada participante de uma VL descoberto em ordem crescente de tempo.

Na seguinte figura 5.3, encontra-se o gráfico dos 10 ensaios da aplicação de teste experimental, mas, neste caso, utilizando-se o sistema de Simple Broadcast. Verifica-se igualmente uma tendência na taxa de crescimento da latência face ao aumento da dimensão da rede, e, consequentemente, ao número de nós source, obtendo-se assim evidências de que essa taxa se manterá para valores mais elevados de escala.

O gráfico da figura 5.4 apresenta uma comparação dos valores máximos de latência entre a utilização do sistema de VLS e o Simple Broadcast, e correspondem aos valores finais dos dois gráficos anteriores 5.2 e 5.3. Pela observação deste gráfico, que relaciona a latência de cada aproximação com o aumento da escala da rede, verifica-se que o sistema de VLS supera a aproximação de disseminação simples, diferença essa que, com uma observação da taxa de crescimento da latência dos dois sistemas, possui tendência para aumentar ainda mais para valores de redes superiores a 81 elementos.

De uma forma mais aproximada, a tabela 5.6 demonstra que o sistema Simple Bro-

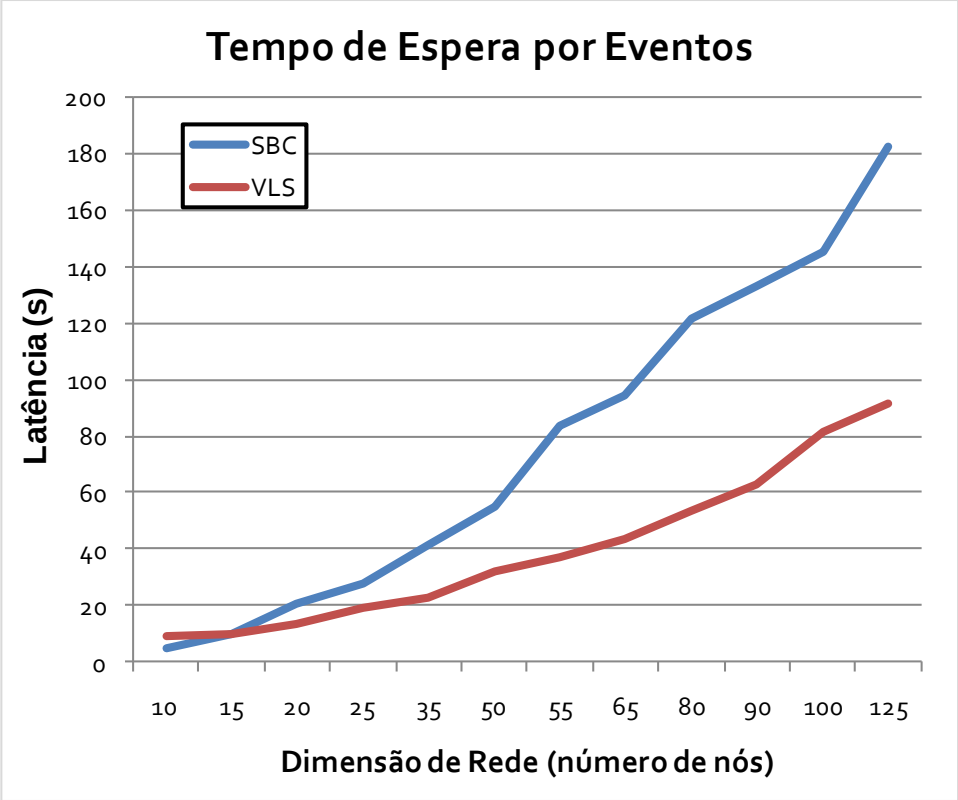


Figura 5.4: Valores médios de latência máxima no processo de criação de uma VL e colecção de um evento, para redes compostas por 9, 25, 49 e 81 elementos.

Núm. Nós	10	15	20	25	35	50	55	65	80	90	100	125
SBC min.	2,8	2,8	2,7	3,8	2,4	1,9	0,9	8,4	1,8	2,5	3,3	2,9
VLS min.	5,7	3,1	5,7	6,2	4,9	2,2	1,3	6,3	4,2	4,2	1,3	4,6
SBC méd.	4,8	9,8	20,6	27,9	41,4	55,1	83,6	94,0	121,5	133,0	144,9	182,5
VLS méd.	9,5	10,2	13,5	19,4	22,6	32,0	37,1	43,9	53,8	62,7	81,2	91,8

Tabela 5.6: Valores médios da latência de criação de uma VLS, em segundos, para os sistemas VLS e SBC, obtidos para as várias dimensões de rede.

adcast consegue obter a primeira resposta à criação de uma vizinhança em menor tempo que a aproximação das VLS. Tal acontecimento é suportado pelo processamento adicional do sistema de VLS, e pela fase de *aging* da cache de dados, que é onde é despoletada a acção de resposta a um interesse. De resto, confirma-se que as VLS conferem um sistema mais rápido em termos de captura de eventos de uma VLS.

Para finalizar esta secção, destaca-se o valor da latência para os últimos valores de dimensão da rede, que, como se verificará no seguinte ponto 5.4.2.2, foi atingido um tecto máximo de tempo expectável de eventos, devido ao facto da fiabilidade não ter sido total para esta dimensão de escala.

5.4.2.2 Avaliação de Indicadores de Fiabilidade

Pretende-se nesta secção analisar se as operacoes sobre espaços de tuplos relacionados com VLS são efectivamente executadas. O principal indicador de sucesso da aplicação de uma operação é evidenciado pela fiabilidade do envio de mensagens na rede. Este indicador é capturado num método em que cada nó source informa o módulo de visualização qual o número de sequência da mensagem que enviou e o nó sink informa a chegada de cada uma.

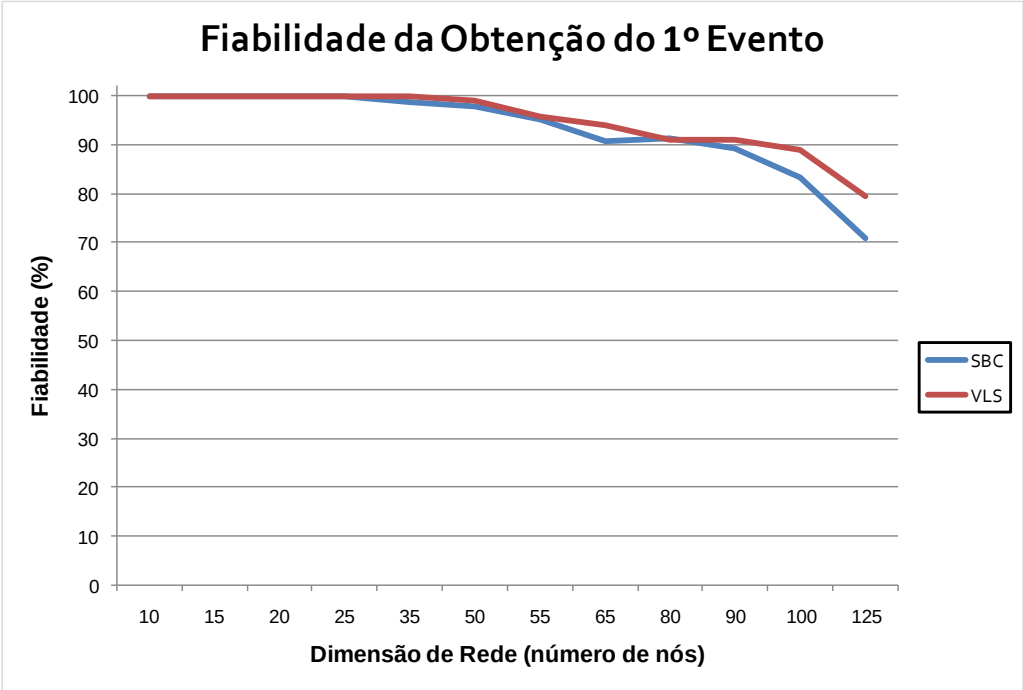


Figura 5.5: Valores médios de latência máxima no processo de criação de uma VL e colecção de um evento, para redes compostas por 9, 25, 49 e 81 elementos.

Por observação ao gráfico 5.5, que relaciona a taxa de fiabilidade das transmissões de eventos capturados com a dimensão da rede, verifica-se que há uma perda de fiabilidade do sistema de Simple Broadcast para escalas acima dos 25 elementos, enquanto que a arquitectura das VLS confere melhores resultados, perdendo fiabilidade apenas acima dos 49 nós. O número de mensagens geradas pelo sistema de Difusão Simples é demasiado elevado, face ao número atingido pelas VLS, o que pode explicar alguma frequência de colisões ao nível link-layer. Note-se que não houve a ocorrência de overflow de buffers, pelo que não se perderam mensagens ao nível do componente de gestão do buffer de envios.

Finalmente, repare-se que as medições dos resultados obtidos no gráfico 5.5 foram realizadas conjuntamente com o mesmo conjunto de testes realizado para o indicador de latência, na secção 5.4.2.1, evidenciando que a perda de fiabilidade resultou no al-

cance do tecto máximo de espera por eventos.

5.4.2.3 Gastos Energéticos

A figura 5.6 ilustra o consumo energético médio de cada nó para cada uma dos sistemas a serem comparados, relacionado com o aumento da dimensão da rede. A primeira observação que se realiza é de que estes valores estão relacionados, de certa forma, com os valores de latência obtidos. Verifica-se, assim, que o sistema de VLS consegue um consumo mais reduzido que a aproximação Simple Broadcast, sendo que tal é explicado, não só com as melhorias introduzidas pelo componente Directed Diffusion, como pelo menor número de mensagens geradas na rede.

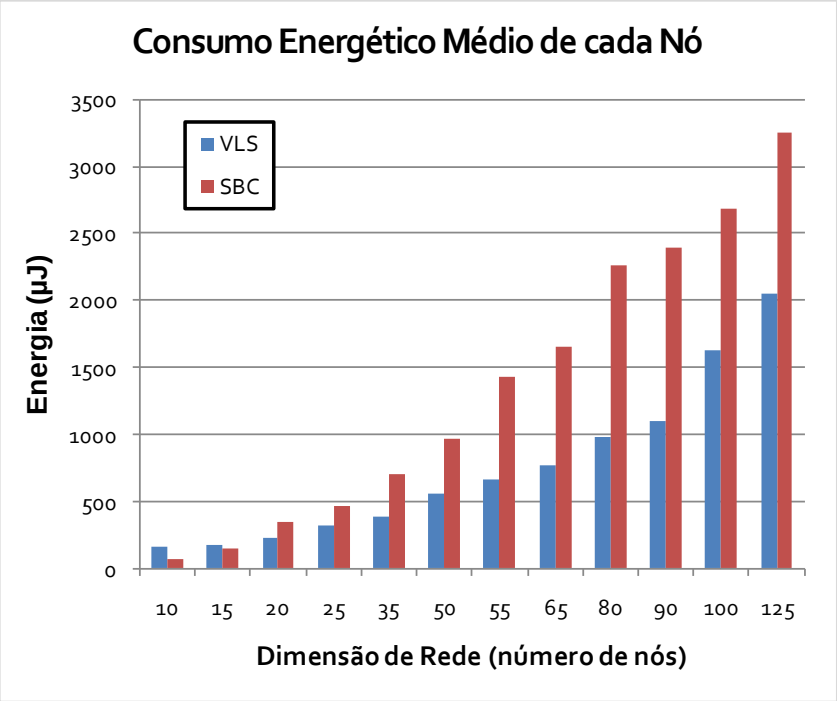


Figura 5.6: Valores médios de consumo energético de cada nó da rede.

O impacto energético médio obtido por dispositivo participante da rede reflecte-se igualmente no valor no consumo total de toda a rede, como se pode observar na figura 5.7.

Relativamente ao impacto no consumo energético de cada dispositivo, outra observação revela essa métrica aplicada apenas sobre os nós source, obtendo-se assim a quantidade de energia dispendida pelo conjunto de nós constituintes de uma VLS. O gráfico na figura 5.8 apresenta um conjunto de valores relativos ao consumo energético de todos os nós que constituem a VLS, perante o consumo de energia concretizado por todos os dispositivos da rede. Note-se que a quantidade de nós pertencentes à VLS

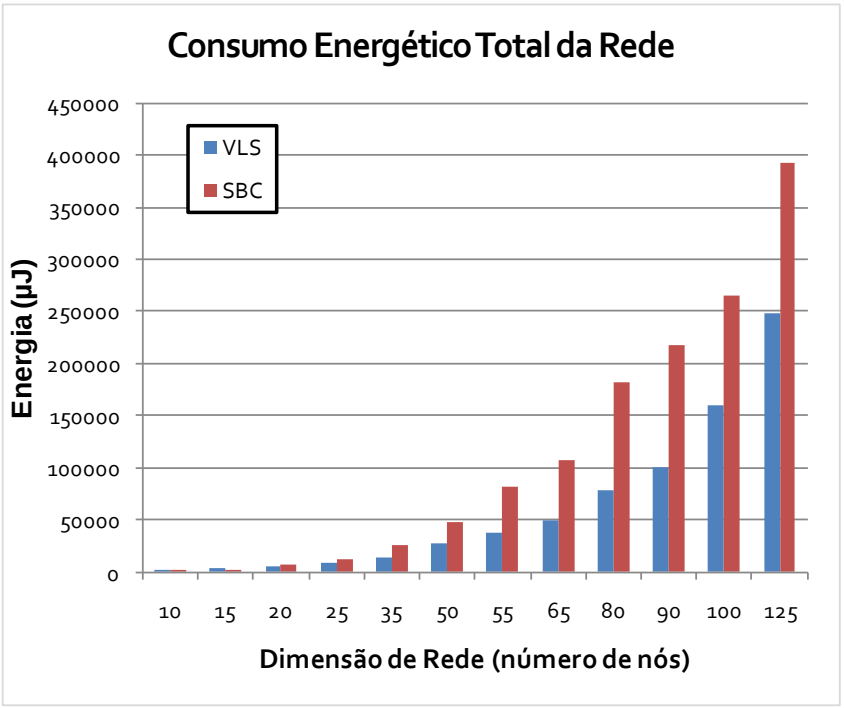


Figura 5.7: Consumo energético total da rede para cada uma das arquitecturas VLS e SBC.

formada na simulação corresponde à metade do número total de nós da rede.

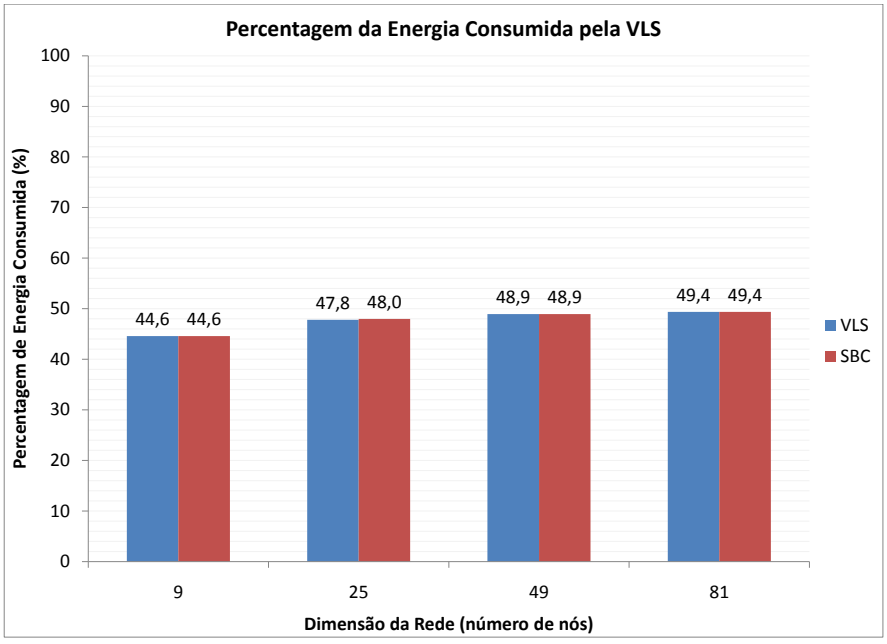


Figura 5.8: Percentagem de consumo energético por parte da VL face ao consumo total da rede.

5.4.2.4 Testes de Estabelecimento de uma VLS com Variação da Percentagem de Participantes

Os testes de criação de uma VLS possuem como um vector determinante a percentagem de participantes de uma rede que faz parte de uma vizinhança lógica. Esse parâmetro de simulação tem como maior incidência na arquitectura ao nível do sistema de interrogação e obtenção de dados, pois determina o número de Sources numa tarefa de pesquisa. Nesta secção, mantém-se a apresentação dos testes incluíndo as métricas quantitativas empregues anteriormente: Fiabilidade na emissão de eventos, tempo de estabelecimento de uma VLS e consumo energético realizado, respectivamente ilustrados nas seguintes figuras 5.11, 5.11 e 5.11.

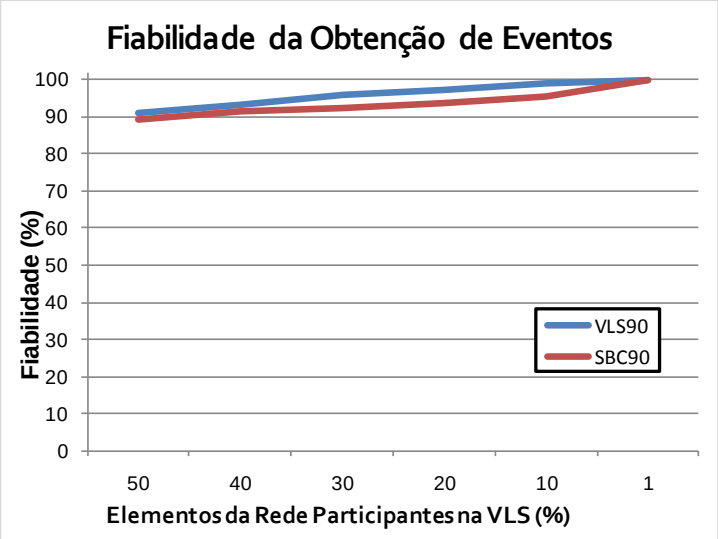


Figura 5.9: Fiabilidade da obtenção de eventos no processo de criação de uma VLS, para várias percentagens de nós Source em relação à dimensão total da rede.

Os valores obtidos revelam uma melhor performance do sistema de VLS, exceptuando no valor de 1%, que representa um caso de referência e onde os valores deveriam ser iguais. No entanto, não o são devido ao *overhead* computacional introduzido pelos sistemas adicionais na arquitectura das VLS, resultando numa maior latência e consumo energético no estabelecimento de uma vizinhança lógica.

5.4.2.5 Testes de Fiabilidade com Variação de Retransmissões

Com a observação ao gráfico da figura 5.12, verifica-se que, com o aumento do número de retransmissões, os níveis de fiabilidade aumentam para cada uma das dimensões de rede. Ao longo do aumento da escala da rede surgem algumas variações nos valores obtidos, relacionadas com algumas diferenças entre as topologias, ao nível

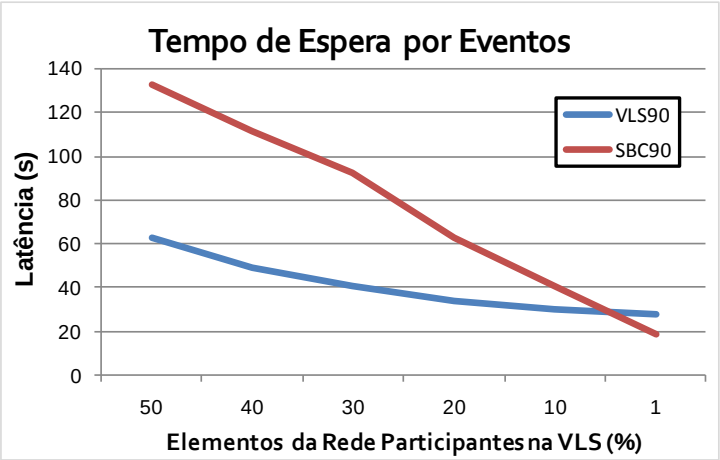


Figura 5.10: Tempo de criação de uma VLS, para várias percentagens de nós Source em relação à dimensão total da rede..

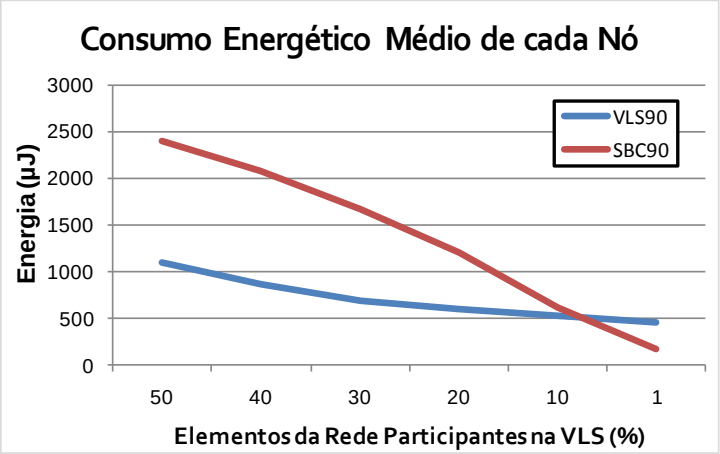


Figura 5.11: Consumo energético realizado no processo de criação de uma VLS, para várias percentagens de nós Source em relação à dimensão total da rede..

do número de Sources existentes em cada distância por saltos do nó Sink. Para redes superiores a 256 nós, as simulações tornam-se demasiado morosas, pelo que se deixa para trabalho futuro a validação experimental orientada para redes de larga escala.

5.5 Síntese Sumária da Validação Experimental

Verificou-se que, pelos testes preliminares, o *middleware* assenta numa plataforma base considerada adequada e confiável (TinyOS + TinySec), cujos componentes principais funcionam de forma disjunta (DD e TL).

Após o estabelecimento de que o MW possui uma base de comunicação que concretiza propriedades de segurança essenciais, como a confidencialidade, autenticidade

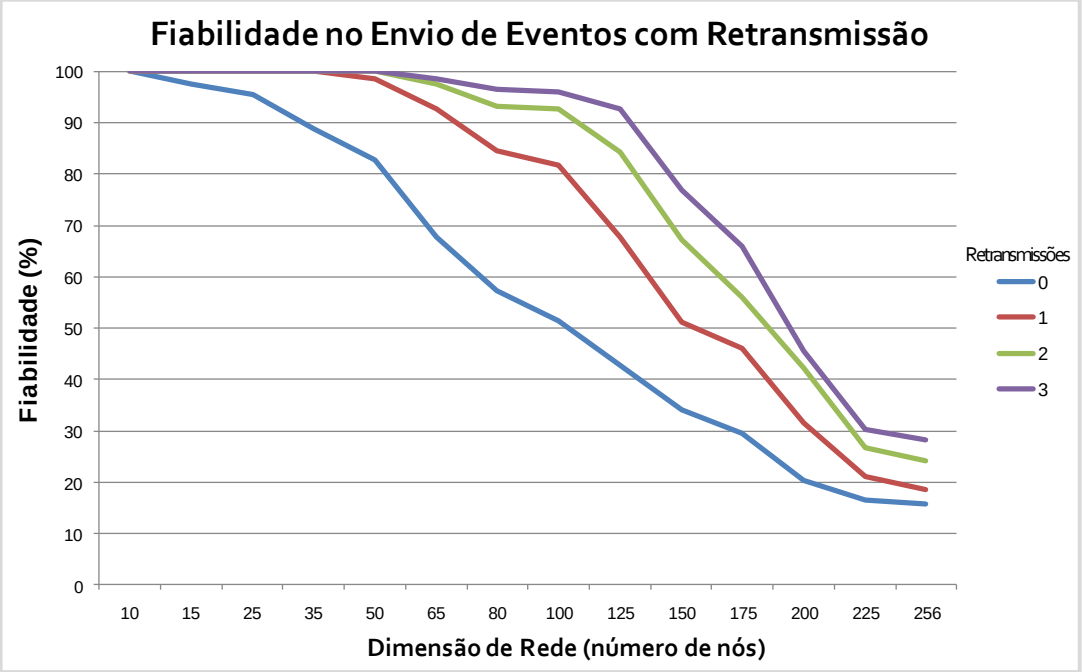


Figura 5.12: Níveis de Fiabilidade do Sistema de VLS para Valores de Retransmissão de 0, 1, 2 e 3.

e integridade dos dados, verificaram-se quais os custos de se possuir os dois sistemas a funcionar de forma combinada, e que anteriormente se encontravam operáveis de forma disjunta. foram realizadas comparações entre tres tipos de sistemas, cada um com as suas caracteristicas próprias: o sistema das VLS, desenvolvido para se obter os objectivos da presente dissertação; e um suporte básico de difusão por *broadcasts* sem organização, Simple Broadcast, considerado como sistema de controlo da comparação.

Verificou-se que o impacto da utilização das VLS é positivo face a um sistema simples de comparação, possuindo uma menor latência na obtenção de eventos, uma maior manutenção da fiabilidade, e um menor consumo de energia, para dimensões comuns ou elevadas de escala da rede. Tal é suportado, essencialmente, pelo menor número de mensagens geradas, nao levando ao incremento dos tempos de espera, nem a colisões, no processo intermédio de encaminhamento.

6

Conclusões

Na presente dissertação, foi proposta e implementada uma arquitectura estratificada de serviços contemplando o paradigma das Vizinhanças Lógicas, noção tornada segura com a incorporação de mecanismos de segurança em vários níveis. Das contribuições principais, integrou-se um serviço de comunicação segura sobre o suporte básico de comunicação rádio, composto por mecanismos de autenticação, confidencialidade, integridade de dados, como criptografia simétrica leve e códigos de autenticação de mensagens. Sobre o serviço base de comunicação segura, colocou-se uma camada de MW composta por: um serviço de pesquisa e difusão de dados escalável, robusto e energéticamente eficiente, com mecanismos de resiliência face a participantes da RSSF que fornecam baixa qualidade de dados; um suporte à abstracção de Vizinhança Lógica Segura, com base na organização da RSSF segundo espaços de tuplos partilhados, definidos por condições definidas ao nível aplicacional.

Pela validação experimental da arquitectura das VLS proposta, obteve-se um sistema com as seguintes contribuições qualitativas: mantém a expressividade de programação das aplicações desenvolvidas para sistemas semelhantes de espaços de tuplos, como o TeenyLIME; acrescenta segurança e fiabilidade ao sistema de espaços de tuplos de referência TeenyLIME, que por sua vez não possui suporte a esses níveis pela razão de ser colocado directamente sobre a camada de rádio. A validação da pilha de componentes edificada foi realizada com a produção de um protótipo experimental, desenvolvido em nesC, sobre o ambiente de desenvolvimento e simulação TinyOS. Os testes ao MW foram focados nas seguintes características quantitativas: latência introduzida na rede; impacto na fiabilidade; e incremento do consumo energético.

Os resultados da validação experimental foram conclusivos, obtendo-se que o Sistema de Vizinhanças Lógicas Seguras demonstrou, face ao sistema original de referência TeenyLIME sobre um suporte de *broadcast* simples, ter um menor impacto: a)

na introdução de latência, b) na degeneração da fiabilidade das comunicações e c) na quantidade de consumo energético total. Com o TOSSIM conseguiu-se reproduzir, com elevada fidelidade, um enorme e demorado conjunto de simulações, mas é um emulador apenas muito próximo da realidade, pelo que não se deve considerar o estudo como um ponto final de avaliação [36].

Esperava-se que o TOSSIM escalasse bem mas como foi observado nos testes efectuados, nao se conseguiram execuções experimentais aceitavelmente fiáveis e conclusivas acima dos cerca de 200 nós. No entanto, repare-se que a complexidade do sistema de VLS é bastante superior à da maioria dos programas desenvolvidos para RSSF, muito mais simples que um sistema MW, pelo que devido à maior carga de processamento e de armazenamento, o TOSSIM já se torna incapaz de simular a utilização do sistema de VLS, com elevada fidelidade, redes de elevada escala.

Sobre a quantificação do impacto da utilização da arquitectura das VLS na validação experimental, e relativamente às métricas de latência, fiabilidade e consumo energético, foram registados valores interessantes de vantagem, em relação a uma arquitectura simples desenvolvida para controlo, obtendo-se, com a utilização da pilha das VLS: diminuição de 25% a 50%, e com melhor escalabilidade evidenciada; perdas de fiabilidade apenas em dimensões de rede superiores a 49 elementos; e ganhos na ordem dos 40% relativamente ao consumo energético.

Desta forma, e tendo em conta as contribuições previstas que se obtiveram com a utilização da arquitectura das VLS, bem como o impacto verificado se ter revelado ter sido significativamente menor em comparação com uma abordagem simples escolhida como sistema de controlo na comparação, pode-se concluir que a utilização do sistema de VLS numa RSSF, onde o desenvolvimento de programação é centrado em torno do paradigma dos espaços de tuplos, contribui com uma elevada vantagem face a abordagens pouco ou nada sofisticadas.

Salientam-se alguns aspectos em aberto que podem surgir na forma de trabalho futuro, interessantes para o melhor funcionamento da arquitectura: o seu desenvolvimento de implementação pode ser concretizado para a versão 2.x do sistema operativo TinyOS, visto a versão empregue na presente dissertação já se encontrar algo obsoleta e pouco eficiente. Caso se pretenda aplicar o *middleware* das VLS, é crucial realizar-se este avanço na versão do sistema operativo alvo.

O presente trabalho teve como alvo as redes de sensores sem fios estáticas em termos de localização. De forma paralela, um passo de evolução passaria por contemplar mobilidade dos nós constituintes da rede, o que aumentaria amplamente as aplicações da tecnologia, como por exemplo, em análise de características de dinâmica de fluídos, entre outras infra-estruturas que exibam mobilidade.

Como foi verificado em [38] que o MiniSec é a melhor proposta ao nível da camada de comunicação segura, seria muito interessante realizar-se a aplicação deste sistema, ao invés do TinySec usado na presente dissertação, que por sua vez não contempla protecção face a ataques por retransmissão. A sua importância ainda viria a demonstrar-se mais conveniente no caso de se realizar a migração para uma versão seguinte do sistema operativo TinyOS. Outra vantagem perspectivada com a utilização do MiniSec compreende um dos seus modos de comunicação: o modo *Broadcast*, que se revelaria com grande utilidade para as difusões completas realizadas na arquitectura das VLS.

Uma melhoria interessante, mas de exigência significativa, passa por aplicar mecanismos de segurança mais específicos ao nível do suporte de pesquisa e difusão de dados do MW das VLS, nomeadamente os propostos no Secure Diffusion, descrito na secção 2.5.4. A inclusão de tais mecanismos é de uma grande ordem de importância, mas não foi coberta pela presente dissertação, visto os níveis de exigência de implementação ultrapassarem os possíveis na realização da presente dissertação.

Mantendo-se a arquitectura de referência inicialmente projectada para a presente dissertação, seria igualmente interessante incluir-se um sistema de estabelecimento e distribuição de chaves secretas, o que, paralelamente com os restantes mecanismos que faltam acrescentar, iria conferir à arquitectura das VLS níveis de segurança bastante elevados.

Uma melhoria ao nível da API de métodos oferecida pela arquitectura aqui concebida pode surgir ao nível da refinação das operações de gestão de VLS. Seria interessante fornecer, ao nível aplicacional, uma capacidade superior de manipulação dos mecanismos de resiliência pró-activa, com indicações de conjuntos de nós preferenciais e conjuntos de nós a evitar, aumentando a sofisticação dos métodos oferecidos ao programador.

Finalmente, seria bastante importante as implementações do suporte a espaços de tuplos e do sistema de pesquisa e difusão de dados serem concluídas. Relativamente ao TeenyLIME, não se contemplaram as operações fiáveis, que obrigam a um maior volume de tráfego na rede, e que podem ter algum impacto significativo nos resultados obtidos. Por outro lado, no Directed Diffusion não se contemplou os processos de agregação, transformação e *nested queries*, que, em grande parte, reduzem o tráfego e o consumo energético [26].

Bibliografia

[1] http://www.sics.se/contiki/.

[2] http://mantis.cs.colorado.edu/tikiwiki/tiki-index.php.

[3] http://www.zigbee.org/.

[4] http://sparrow.ece.cmu.edu/group/minisec.html.

[5] http://teenylime.sourceforge.net/.

[6] http://sourceforge.net/apps/mediawiki/teenylime/index.php?title=Getting_started.

[7] http://docs.tinyos.net/index.php/Running_a_XubunTOS_Virtual_Machine_Image_in_VMware_Player.

[8] http://gems.leme.org.pt/PmWiki/index.php/Resources/HowToInstallTinyOS.

[9] http://nescc.sourceforge.net/.

[10] IEEE Std 802.15.4. Standard for part 15.4: Wireless medium access control (mac) and physical layer (phy) specifications for low rate wireless personal area networks (wpan). New York, 2003. IEEE.

[11] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38:393–422, 2002.

[12] Lan F. Akyildiz, Welljan Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. A survey on sensor networks, 2002.

[13] ZigBee Alliance. Zigbee specification, technical report document 053474r17. San Ramon, CA 94583, 2008. ZigBee Alliance, Inc.

[14] Archana Bharathidasan, Vijay An, and Sai Ponduru. Sensor networks: An overview. Technical report, Department of Computer Science, University of California, Davis, 2002.

[15] D. Boyle and T. Newe. Security protocols for use with wireless sensor networks: A survey of security architectures. In *ICWMC '07: Proceedings of the Third International Conference on Wireless and Mobile Communications*, page 54, Washington, DC, USA, 2007. IEEE Computer Society.

[16] Haowen Chan, Adrian Perrig, and Dawn Song. Random key predistribution schemes for sensor networks. In *SP '03: Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 197–213, Washington, DC, USA, 2003. IEEE Computer Society.

[17] Yih chun Hu, Adrian Perrig, and David B. Johnson. Packet leashes: A defense against wormhole attacks in wireless networks. 2001.

[18] Paolo Costa, Luca Mottola, Amy L. Murphy, and Gian Pietro Picco. Teenylime: transiently shared tuple space middleware for wireless sensor networks. In *MidSens '06: Proceedings of the international workshop on Middleware for sensor networks*, pages 43–48, New York, NY, USA, 2006. ACM.

[19] Jing Deng, Richard Han, and Shivakant Mishra. Insens: Intrusion-tolerant routing for wireless sensor networks. *Computer Communications*, 29(2):216–230, January 2006.

[20] Laurent Eschenauer and Virgil D. Gligor. A key-management scheme for distributed sensor networks. In *CCS '02: Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 41–47, New York, NY, USA, 2002. ACM.

[21] Chien-Liang Fok, Gruia-Catalin Roman, and Chenyang Lu. Mobile agent middleware for sensor networks: An application case study. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 51, Piscataway, NJ, USA, 2005. IEEE Press.

[22] David Gay, Matt Welsh, Philip Levis, Eric Brewer, Robert von Behren, and David Culler. The nesc language: A holistic approach to networked embedded systems. In *In Proceedings of Programming Language Design and Implementation (PLDI*, pages 1–11, 2003.

[23] David Gelernter. Generative communication in linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, 1985.

[24] John Heidemann, Fabio Silva, Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, and Deepak Ganesan. Building efficient wireless sensor networks with low-level naming, 2001.

[25] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. *Hawaii International Conference on System Sciences*, 8:8020, 2000.

[26] Chalermek Intanagonwiwat. *Directed Diffusion: An Application-Specific and Data-Centric Communication Paradigm for Wireless Sensor Networks*. PhD thesis, 2002.

[27] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *MOBICOM*, pages 56–67. ACM, 2000.

[28] Martin Jansson. Context shadow: An infrastructure for context aware computing. In *Third workshop on Artificial Intelligence in Mobile Systems - AIMS*, 2002.

[29] Yixin Jiang, Chuang Lin, Minghui Shi, and Xuemin (Sherman) Shen. Key management schemes for wireless sensor networks. In *Security in Sensor Networks, Edited by Yang Xiao*, pages 113–143, Boca Raton, FL, USA, 2007. Auerbach Publications, Taylor and Francis Group.

[30] C. Karlof and D. Wagner. Secure routing in wireless sensor networks: attacks and countermeasures. In *Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE. 2003 IEEE International Workshop on*, pages 113–127, May 2003.

[31] Chris Karlof, Naveen Sastry, and David Wagner. Tinysec: User manual. `webs.cs.berkeley.edu/tos/tinyos-1.x/doc/tinysec.pdf`.

[32] Chris Karlof, Naveen Sastry, and David Wagner. Tinysec: a link layer security architecture for wireless sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 162–175, New York, NY, USA, 2004. ACM Press.

[33] Mauri Kuorilehto, Marko Hännikäinen, and Timo D. Hämäläinen. A survey of application distribution in wireless sensor networks. *EURASIP J. Wirel. Commun. Netw.*, 2005(5):774–788, 2005.

[34] Nelson Lee, Philip Levis, and Jason Hill. Mica high speed radio stack, 2002.

[35] Philip Levis. Tep 111: message t. `http://www.tinyos.net/tinyos-2.x/doc/pdf/tep111.pdf`.

[36] Philip Levis and Nelson Lee. Tossim: A simulator for tinyos networks, 2003. `www.cs.berkeley.edu/~pal/pubs/nido.pdf`.

[37] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. Tossim: accurate and scalable simulation of entire tinyos applications. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 126–137, New York, NY, USA, 2003. ACM.

[38] Mark Luk, Ghita Mezzour, Adrian Perrig, and Virgil Gligor. Minisec: a secure sensor network communication architecture. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 479–488, New York, NY, USA, 2007. ACM.

[39] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[40] David J. Malan, Matt Welsh, and Michael D. Smith. A public-key infrastructure for key distribution in tinyos based on elliptic curve cryptography, 2004.

[41] Vojislav B. Misic, Jung Fung, and Jelena Misic. Mac layer attacks in 802.15.4 sensor networks. In *Security in Sensor Networks, Edited by Yang Xiao*, pages 27–44, Boca Raton, FL, USA, 2007. Auerbach Publications, Taylor and Francis Group.

[42] Luca Mottola and Gian Pietro Picco. Logical neighborhoods: A programming abstraction for wireless sensor networks. In *Proceedings of the 2nd International Conference on Distributed Computing on Sensor Systems (DCOSS)*, 2006.

[43] Amy L. Murphy, Gian Pietro Picco, and Gruia-Catalin Roman. Lime: A middleware for physical and logical mobility. pages 05–24, Los Alamitos, CA, USA, 2001. IEEE Computer Society.

[44] James Newsome, Elaine Shi, Dawn Song, and Adrian Perrig. The sybil attack in sensor networks: Analysis and defenses, 2004.

[45] Leonardo B. Oliveira, Hao C. Wong, M. Bern, Ricardo Dahab, and A. A. F. Loureiro. Secleach - a random key distribution solution for securing clustered sensor networks. In *NCA '06: Proceedings of the Fifth IEEE International Symposium on Network Computing and Applications*, pages 145–154. IEEE Computer Society, 2006.

[46] Bryan Parno, Mark Luk, Evan Gaustad, and Adrian Perrig. Secure sensor network routing: a clean-slate approach. In *CoNEXT '06: Proceedings of the 2006 ACM CoNEXT conference*, pages 1–13. ACM, 2006.

[47] Adrian Perrig, Ran Canetti, J. D. Tygar, and Dawn Song. The tesla broadcast authentication protocol. *RSA CryptoBytes*, 5:2–13, 2002.

[48] Adrian Perrig, John Stankovic, David Wagner, and Caren Rosenblatt. Security in wireless sensor networks. *Communications of the ACM*, 47:53–57, 2004.

[49] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and J. D. Tygar. Spins: Security protocols for sensor networks. In *Wireless Networks*, pages 189–199, 2001.

[50] V. Rajaravivarma, Yi Yang, and Teng Yang. An overview of wireless sensor network and applications. In *System Theory, 2003. Proceedings of the 35th Southeastern Symposium on*, pages 432–436, 2003.

[51] Victor Shnayder, Mark Hempstead, Bor rong Chen, Geoff Werner Allen, and Matt Welsh. Simulating the power consumption of large-scale sensor network applications. In *In Sensys*, pages 188–200. ACM Press, 2004.

[52] Victor Shnayder, Mark Hempstead, Bor rong Chen, and Matt Welsh. Powertossim: Efficient power simulation for tinyos applications. `http://www.eecs.harvard.edu/~shnayder/ptossim/`.

[53] William Stallings. *Data & Computer Communications*. Prentice-Hall, Englewood Cliffs, NJ, USA, 2001.

[54] IEEE 802.15 WPAN Task Group 4 (TG4). `http://www.ieee802.org/15/pub/TG4.html`.

[55] You-Chiun Wang and Yu-Chee Tseng. Attacks and defenses of routing mechanisms in ad hoc and sensor networks. In *Security in Sensor Networks, Edited by Yang Xiao*, pages 3–25, Boca Raton, FL, USA, 2007. Auerbach Publications, Taylor and Francis Group.

[56] Matt Welsh and Geoff Mainland. Programming sensor networks using abstract regions. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 29–42, Berkeley, CA, USA, 2004. USENIX Association.

[57] Kamin Whitehouse, Cory Sharp, David E. Culler, and Eric A. Brewer. Hood: A neighborhood abstraction for sensor networks. In *MobiSys*, 2004.

[58] Hao Yang, Starsky H. Y. Wong, Songwu Lu, and Lixia Zhang. Secure diffusion for wireless sensor networks. In *BROADNETS*, pages 1–10, 2006.

[59] H. Zimmermann. Osi reference model–the iso model of architecture for open systems interconnection. *Communications, IEEE Transactions on [legacy, pre - 1988]*, 28(4):425–432, 1980.